The University of Alberta
Printing Department
Edmonton, Alberta

UNIVERSITY OF ALBERTA
LIBRARY

Regulations Regarding Theses and Dissertations

Typescript copies of theses and dissertations for Master's and Doctor's degrees deposited in the University of Alberta Library, as the official Copy of the Faculty of Graduate Studies, may be consulted in the Reference Reading Room only.

A second copy is on deposit in the Department under whose supervision the work was done. Some Departments are willing to loan their copy to libraries, through the inter-library loan service of the University of Alberta Library.

These theses and dissertations are to be used only with due regard to the rights of the author. Written permission of the author and of the Department must be obtained through the University of Alberta Library when extended passages are copied. When permission has been granted, acknowledgement must appear in the published work.

This thesis or dissertation has been used in accordance with the above regulations by the persons listed below. The borrowing library is obligated to secure the signature of each user.

Please sign below:

Date	Signature	Institution

THE UNIVERSITY OF ALBERTA

COMPUTER GRAPHICS IN LOGIC CIRCUIT DESIGN

by

© Brian Victor Johnson

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE

OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTING SCIENCE

EDMONTON, ALBERTA

FALL, 1969

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled COMPUTER GRAPHICS IN LOGIC CIRCUIT DESIGN submitted by Brian Victor Johnson in partial fulfillment of the requirements for the degree of Master of Science.

ABSTRACT

An investigation has been made into the problems of building and implementing computer-aided circuit design systems using a graphical display. A survey of the field has been presented and a proposal for a computer-aided logic design system (CALD) put forth. The implementation of CALD has been specified in detail and some of the fundamental difficulties in building systems such as CALD have been discussed. Further extensions and improvements for CALD have been suggested and the possibility of a syntax-directed system has been considered.

ACKNOWLEDGEMENTS

I express my appreciation to Professor J.P. Penny for his advice, criticism, and guidance throughout the duration of this research. I am indebted to the programming support of Wing Hing Huen, Frank Jacobsen, and Ken May. Special thanks are extended to my wife Barbara for her assistance in the initial typing and proofreading and to Betty Florchyk for typing the final draft.

I wish to thank the Department of Computing Science for the financial assistance and facilities which have made this project possible.

TABLE OF CONTENTS

LIST OF FIGURES

...the desire for interaction between man and machine is based on a fundamental mistrust of the machine left alone to its pre-programmed devices.

- Unknown

CHAPTER I

INTRODUCTION

> "Communicating through the medium of the visual display adds a new dimension to computer technology" (Chasen 1968)

> "Computer Graphics represents a new stage in the art of visual communication... Through Computer Graphics visual communication has developed into a technology, with computers converting virtually unlimited engineering or scientific data into visual images" (Fetter 1968)

> "The enormous interest that graphics has generated certainly at the present time, is all out of proportion to any positive yield that it has as yet returned" (Cancro 1968)

> "Computer aided design has been something less than a smash hit during its comparatively brief run... CAD has clearly been puttering away at relatively unsophisticated tasks. (Mittleman 1969)

The excitement and optimism generated in the first two quotations is certainly dampened somewhat by the last two remarks. It is generally accepted that computer graphics offers tremendous potential for man-computer interaction in such areas as computer-aided design. However, it appears that much of this potential has not been realized and the success of computer-aided design with computer graphics has been less than tremendous. Cancro et al. (Cancro et al. 1968) suggest that the purpose of graphics is to alter meaningfully and positively the dialogue between man and

machine so that the working relationship between them will be more productive. Surely then, computer graphics is suited to computer-aided design.

Mittleman (Mittleman 1969) states:

"Computer-aided design, though growing up, has yet to come of age. Programming, graphics, and modeling are among the problems that must still be solved before the computer's potential -- the synthesizing and optimizing of designs -- can be fully realized".

Obviously, the obstacles in achieving the promoted potential of computer graphics have been of greater magnitude than first anticipated. Out of this realization has come the following investigation: this thesis is concerned with the designing and building of a computer-aided design system, more specifically a computer-aided circuit design system, using a graphical display. This investigation includes a project to build and implement a computer-aided logic design system (CALD). The major objective of this project is to obtain a better understanding of what is involved in building systems of this nature.

A survey of the history of computer-aided design is presented from the initial batch processing techniques to the present methods using on-line graphical displays. The requirements of design systems are presented and the problems in implementing systems of this nature are discussed.

Two circuit design projects, OLCA (So 1966) and LDB

(Cross 1967) are described in detail. A comparison is made of the two systems and they are evaluated in terms of the system requirements which had been previously set out. These projects serve as a guide in the design of CALD.

CALD is then presented in detail. Firstly, a proposal for CALD is given, from its requirements to the first version objectives. Then the CALD system is described in detail in terms of design technique, program structure, and data structure. CALD is evaluated and suggestions are made for improvements and extensions to the present version.

Out of the investigation in building a system of the nature of CALD evolved some thoughts for future research. The possibility of building a syntax directed system is suggested and the concepts of such a system are discussed.

CHAPTER II

SURVEY OF DESIGN

2.1 Introduction

Electronic circuit design, more especially digital computer design, involves three basic steps: system design, circuit design, and logical design (Phister 1960). In designing a computer or a component for a computer, the systems design group defines and analyzes the operation to be performed by the required device. They then embody a set of specifications which describe in detail what the machine should do and how. The circuit design group takes a set of basic elements (resistors, capacitors, transistors) and connects them together in circuits which will perform well-defined operations. The third group, the logical design group, takes the specifications set out by the systems group, and the components perfected by the circuit design group, and produces a set of wiring diagrams which show how to fit the components together to realize the specifications. Though computer design is not necessarily done by three groups of people, the three design steps will be inherent in their design process.

It is the purpose of this chapter to present a broad view of the history and the state of the art of computer-aided design, with primary interest in on-line design, particularly using graphical displays. Following in Chapter III is a detailed description of two computer-aided

design systems.

In much of the literature which discusses logical design, the authors use the term circuit design in the same sense as the term logical design has just been defined; therefore to conform to the conventional terminology, circuit design and logic design will be used interchangeably.

2.2 Batch Circuit Analysis

2.2.1 Conventional Approach

There are two fundamental problems in the theory of circuit design (Shalla 1966, Evans et al. 1967): synthesis and analysis. Synthesis involves taking a description of the required function of a device and building a minimal circuit which performs this function. Analysis involves taking a given circuit and describing (by testing) what function it performs.

Dertouzos (Dertouzos-1 1967) describes a typical circuit design process in Figure 2.1.

In conventional circuit design, without the assistance of a computer, the development phase of a circuit may last weeks or months. With the advent of the computer, and algorithmic mathematics applied to circuit analysis (Dertouzos-2 1967), it is now possible to use the computer to assist in performing circuit analysis (Kuo-1 1966, Wall 1964, Christiansen 1966, Sedore 1966, Ninke 1965). In batch computer-aided circuit design, the downflow of

Figure 2.1 is accelerated since the use of the computer is engaged for the analysis task. The analysis is more accurate and, because of the versatility of the computer analysis, the number of prototype tests is reduced. The development phase is commonly decreased by a factor of six to eight.

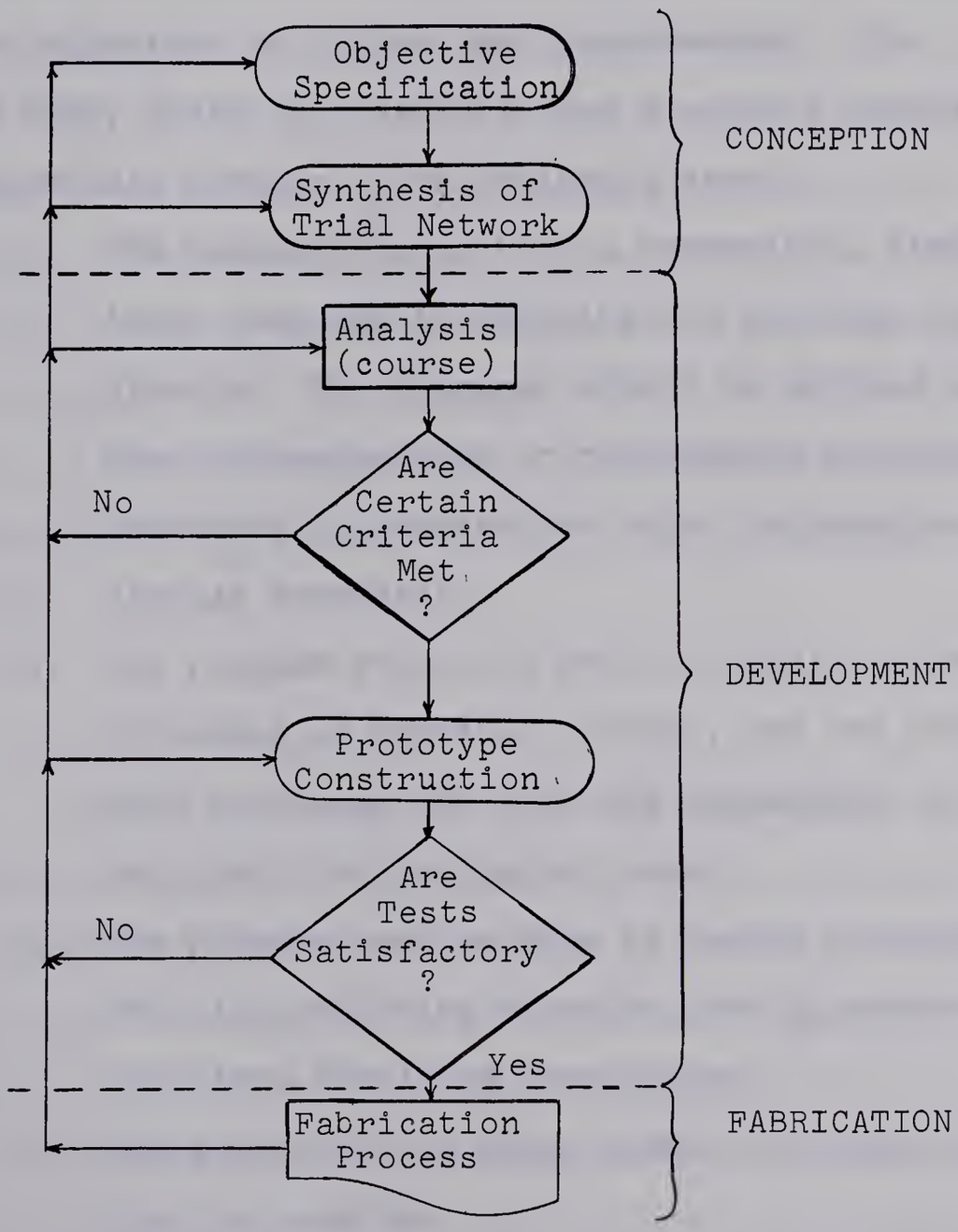


Figure 2.1 Typical Design Process

2.2.2 Circuit Analysis Program Requirements

Consider what should be the requirements of computer-aided circuit analysis programs. Sedore (Sedore 1966) suggests that "an automatic circuit analysis program can be defined as one that does not require the user to write any circuit equations or do any real programming". Kuo (Kuo-1 1966) lists six features that a network analysis program should contain to be maximally useful:

1. The program should have a convenient, simple input language to describe the topology of the circuit. The language should be devised such that no engineering or programming knowledge is necessary to prepare the input information from a circuit schematic.
2. The program should be able to handle a wide class of models of physical devices, and one should be able to change not just the parameters of a device but also its topological model.
3. The program must be able to handle nonlinearities, that is, switching circuits, and quiescent solutions involving transistors.
4. There should be a large number of output options for the program.
5. The program should facilitate automatic parameter modifications to enable sensitivity, tolerance, and worst case studies to be performed. (This

might drastically reduce the number of prototype construction steps in Figure 2.1).

6. The program should contain error checks so that the user can assess the reliability of the input.

The intent at this point is not to investigate circuit analysis programs; the reader, however, is referred to the work of Dawson et al. (Dawson et al. 1967) who have examined a good number of electronic circuit analysis programs from a standpoint of the user and have tabulated the major features and capabilities of the following well known programs: NET-1, SCEPTRE, CIRCUS, CALAHAN, ECAP, POTTLE, and PREDICT. Kuo (Kuo-1 1966) also discusses some network analysis programs with particular attention to NET-1 and ECAP.

2.2.3 Drawbacks

While batch computer-aided circuit design has been valuable in speeding up analysis and making trial and error and various optimization procedures practicable (Kuo-1 1966), there are a number of drawbacks to this type of design which make it less than ideal. Generally the circuit analysis programs do not satisfy the first requirement of analysis programs which was stated above. The coding required to translate the circuit into acceptable input for the computer is usually according to rigidly specified formats and is liable to be tedious and error-prone. This

type of input is alien to the design engineer, especially if the designer must spend half his time debugging invalid input data. Also, while the time for computer analysis is in the order of seconds or minutes the turnaround time for the design engineer in a typical batch environment is in the order of several hours.

The objection to this type of computer-aided circuit design has been summed up simply by So (So 1967) as "difficult interaction between man and machine". It is desirable for the engineer to specify his problem naturally without contending with complex data formats and it is desirable for a designer to obtain his results "instantly" so that the design experimentation can be carried out in a smooth uninterrupted fashion.

2.3 On-Line Circuit Design

2.3.1 What is on-line circuit design?

Dertouzos (Dertouzos-2 1967) has introduced a meaning for on-line circuit design: the same meaning is adopted here. Much of the following discussion in this section is taken from the work of Dertouzos. Therefore the continued repetition of referencing his work will be omitted). He describes what on-line circuit design is by illustrating a typical design process. This typical process consists of a circuit designer seated in front of a graphical display console equipped with light pen and function buttons. The circuit

schematic is constructed by pointing with the light pen at the screen and by pressing push (or light) buttons corresponding to appropriate circuit elements. These elements will then appear at the points selected by the light pen. After a circuit has been constructed, the user can establish voltages and currents; he may then ask for results such as voltage versus time plots. The designer will then inspect his results and, using the light pen, he will undoubtedly modify the circuit by detaching elements, adding elements, or changing their values. Dertouzos states

> "If the designer and the machine can conduct such a design dialogue with short interaction delays (on the order of seconds) then we refer to the process as on-line circuit design."

On-line circuit design suggests a possible answer to the problems or objections to batch oriented circuit design. On-line circuit design can facilitate fast man-machine interaction (Dertouzos-2 1967, Spitalny et al. 1967, So 1966). Also on-line circuit analysis allows the designer to obtain intermediate results in the circuit analysis and perhaps exercise some control of the program at decision points. In Figure 2.1, on-line circuit design can not just speed the downflow but also can accelerate the upflow in this type of design loop.

2.3.2 On-line Utility

The prerequisite for on-line circuit design is an on-

line digital computer utility. The utility consists of computer hardware and supervisory software. The hardware components are processor(s), memory(ies), and input/output equipment. The software supervisor is responsible for hardware management, resource allocation, and user access control.

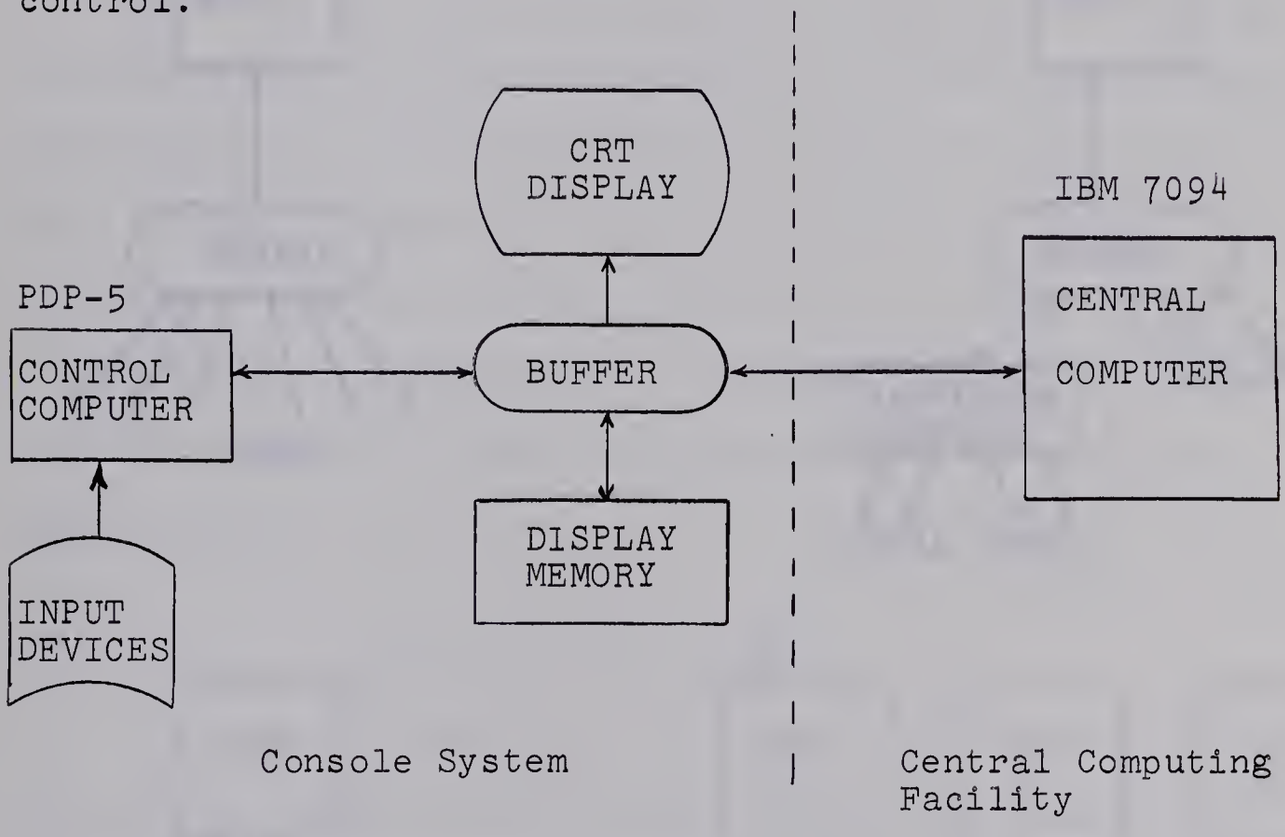


Figure 2.2 OLCA Configuration (So 1966)

The on-line utility can be constructed in various hardware configurations; each configuration because of its uniqueness will require its own special software supervisor. A typical operating environment, presented by So (So 1966), is OLCA, Figure 2.2, which operates with a central processor interfaced with a satellite computer. Other configurations,

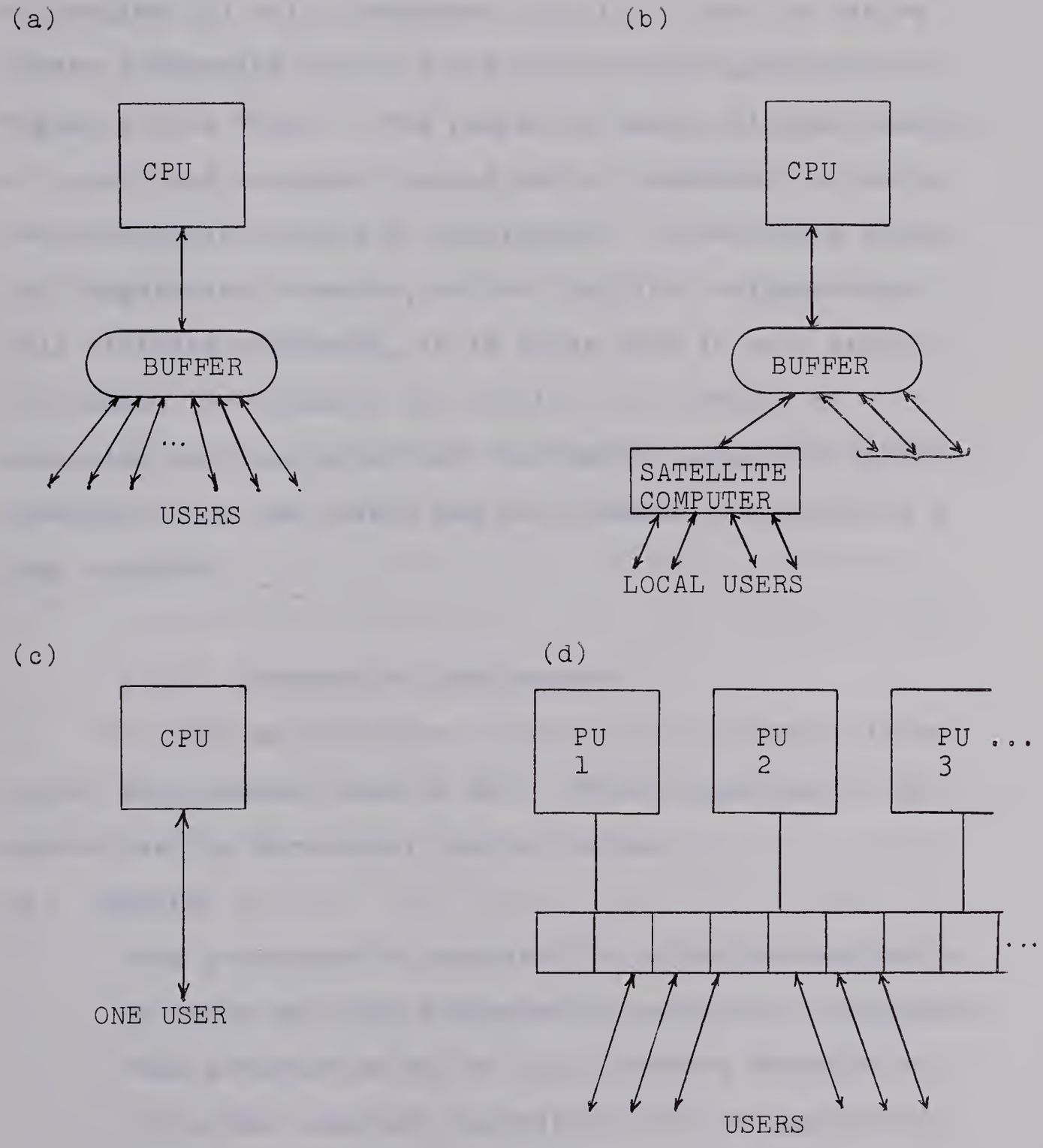


Figure 2.3 On-line Utility Configurations

Figure 2.3 (Dertouzos-2 1967) exist: (a) single time shared processor (b) satellite computer (c) batch converted to on-line (d) multi-processor utility. There is active debate concerning which of the utility configurations in Figure 2.3 is "best". The number of users, diversification of users, and economic factors are all relevant to how an on-line utility should be configured. As Dertouzos points out "Regardless, however, of what utility configuration will ultimately persist, it is clear that it will strive to present the illusion (or reality) of a wealth of computing services which are "instantly" under the user's command, as if the latter had the constant attention of a huge computer".

2.3.3 Interaction Requirements

To have an effective on-line utility certain interaction requirements must be met. These requirements, as generalized by Dertouzos, are as follows:

(a) Editing

This provision is necessary to allow information to be input and then subsequently modified if required. This information may be (1) a network description (2) a new component description (3) an analytical, graphical or tabular representation of a function. These editing functions are often accomplished through the use of a teletypewriter, a graphical console,

push buttons, light buttons, or a combination of these devices.

(b) Output

This provides for the direct output of input information or the output of computed data. This output can be the display of network schematics, function graphs, or element descriptions. The medium for output can be the graphical display, typewriters, or digital plotters.

(c) Definitional

This provision permits a program to dynamically grow, develop, and adapt to prevailing demands. This ability may be exercised by creating "new" design components or by creating "new" edit commands which can be used in the system.

(d) Informational

This allows the user to request information concerning the design process in which he is presently engaged. He may request such items as "what have I done so far?"; "describe the functional representation of component X"; or the designer may request information regarding the use of the system, that is, "what do I do next?".

(e) Diagnostics

This provision is part of a safeguard system. In a complex system, the computer must examine the user

actions to keep to a minimum the mistakes which will confuse the designer and/or the system. Classes of errors that a diagnostic ability can detect are usually errors of form or format and user violation of problem-oriented "laws".

Of the requirements listed above perhaps the most powerful and potentially dangerous provision is the existence of a definitional ability. If you allow a system to exercise dynamic growth, then it is difficult to constrain the system (diagnostic requirement) such that you prevent the designer from "blowing" the system and yet still allow him flexibility in defining his tasks.

A users view of an on-line circuit design program has been represented in Figure 2.4.

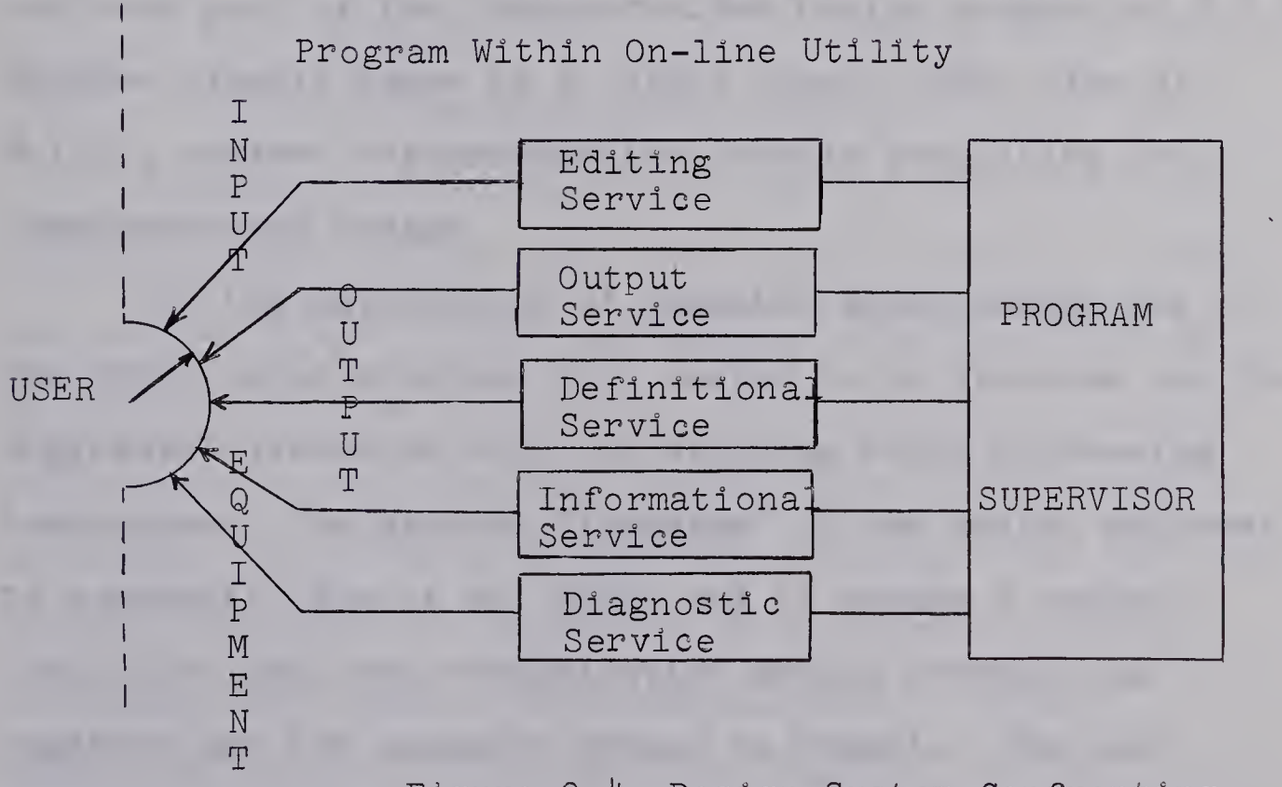


Figure 2.4 Design System Confronting User (Dertouzos-2 1967)

2.4 On-Line Graphics

2.4.1 History

Prince (Prince 1966) has given a comprehensive paper on man-computer graphics, the concepts and state of the art; much of what is presented below is taken from his paper.

I. Sutherland in his "Sketchpad" system on the TX-2 computer at Lincoln Laboratory (Sutherland 1963) made the first step in initiating graphical computer-aided design. A second important step was taken by T.E. Johnson with "Sketchpad III" (Johnson 1963) in which he presented a computer program for drawing in three dimensions. The impact of graphical computer-aided design was realized in papers by D.T. Ross (Ross 1963) and S.A. Coons (Coons 1963) who were part of the Computer-Aided Design program at M.I.T. Another classic paper by R. Stotz (Stotz 1963) also of M.I.T., reviews the man-machine console facilities for computer-aided design.

In the development of computer-aided design one of the first major problems that needed to be overcome was the engineers alienation with the existing batch processing techniques. The natural "language" of the design engineer is schematic (Kuo et al. 1969) and it became a natural conclusion that the communication medium between the engineer and the computer should be graphic. On-line graphical output for on-line design is a very convenient

mode of communication between man and machine (Murray-Lasso 1968). The designer can examine the effect of a parameter change on the circuit response in everyday terms without having to translate his objectives into precise mathematical formulations.

The circuit designers soon became aware of the potential of graphics, and out of this realization sprung a multitude of systems and programs, some of which are described in Cross 1967, Dertouzos-1 1967, Fisk et al. 1967, Kuo et al. 1969, Murray-Lasso 1968, Ninke-1 1965, Ninke-2 1966, Prince 1966, So-1 1966, So-2 1967, Spitalney et al. 1967, Evans et al. 1967, Parker 1967. These systems were built for electronic design using computer graphics.

2.4.2 Hardware

2.4.2.1 Computer-Driven Display

Commonly, the display is a cathode ray tube with a 12 x 12 inch display area. The display area is usually composed of 1,024 x 1,024 addressable locations. Points and symbols may be displayed at any addressable location and vectors can be displayed between any two addressable locations. The CRT is usually not a storage tube and so a given picture must be continuously regenerated to produce a flicker-free image on the tube phosphor. Typical repetition rates are from 30 to 60 times a second.

2.4.2.2 Light Pen

The light pen (Ninke-2 1966) is a hand-held, pen-size sensing device; it is a fiber-optic bundle which terminates in a photomultiplier tube; a shutter is opened when the button on the pen is depressed. The light pen photomultiplier sends a pulse (interrupt) to the computer at the instant that the light pen "sees" some light on the screen. The computer "knows" which line, symbol, or point was "seen" by the light pen from the address of the element in the computer. By proper programming, symbols or labels on the screen can be made light pen sensitive such that the user can "pick" (select) components on the screen.

2.4.2.3 Alpha-numeric keyboard and function buttons

The keyboard and function buttons are manual input devices which, when activated, provide a unique code (interrupt) to be interpreted by the computer. The actions to be followed by the computer, when some or any of these keys are depressed, can be controlled by the system programmer.

2.4.3 Levels of Graphic Utilization

2.4.3.1 Output

The least sophisticated level of graphic utilization is using the display only as an output device to display pre-stored data in the form of graphs, tables,

network and structural diagrams, density patterns, geometric patterns, etc. This level of graphics can be used for real-time observation of a computer problem program output.

2.4.3.2 Input and Output

The next level of graphic utilization, as well as making use of the output facility, utilizes input devices such as light pen, alpha-numeric keyboards, and function buttons. Generally the user can exercise some type of control over the display program by creating program interrupts which the system decodes and then executes an appropriate service routine for the interrupt.

2.4.3.3 Topological

The highest level of graphic utilization and of course the most sophisticated, requires the computer program to "know" more than just the coordinates of the displayed picture. The program is required to have some "knowledge" of the topology of the picture being displayed, that is to say, that besides being able to display the picture the program must have some knowledge of what the picture represents.

This is the level of utilization at which circuit design is done, for clearly, if a program is going to analyze a circuit effectively, then it will be necessary for it to store more information about the circuit than just the picture schematics.

2.5 Program (Data) Structures

A data structure (Parker 1967) is the context into which data is placed for storage and use by the computer program. It is concerned with format, ordering, and interconnection or association of data.

The first type of data structure that was used was of course, the simplest: it consisted of storing of information in contiguous locations in memory, usually in the form of one or two dimensional arrays. The need for more complex structures became apparent with the development of compilers which required text manipulation and the parsing of statements in programming languages. From this need, symbol-manipulating and list-processing languages (Bobrow 1964) developed. Sutherland (Sutherland 1963) and Ross (Ross 1963) showed that the list structures of the list-processing systems were not adequate for graphic data representations because they lacked the multi-dimensional and variable block size schemes.

Ninke (Ninke-2 1966) argued that the central requirement of interactive work on a display console is a computer stored data structure which incorporates, in addition to individual parameters or data attached to a problem entity, the associations or inter-relations among the various entities of the system. The console user must have facilities to compose the data structure initially, to manipulate and change it, to obtain visual feedback from it,

and to do processing or further computation using it.

In an operational environment, there exist in the system two "languages": the application program language and the console command language. The program language is usually a high level programming language (not involving machine coding) in which the application program can build and present its system using general statements similar to a natural language. The application program language support will automatically translate the statements into appropriate machine instructions. All the programming required for data structure building and manipulation, and application problem analysis is done in the application program language.

The console command language is the interface between the user and the system (application program). The user and the applications program perform their dialogue through the console command language. The designer specifies his console commands through ordered manipulations of the console input devices. The actions of the console command language generally are translated into data structure modifications or manipulations. Visual feedback is then produced by analyzing the data structure to produce a pictorial representation of the contents and structure of the data.

In circuit design, by means of the console command language, the functional model of the circuit and its

schematics can be specified in the data structure. The data structure can then be used as input to a circuit analysis program.

The basic or generic unit of data in a data structure is called a block, item, or entity (Parker 1967). The inter-relations or associations between the blocks or entities are represented by a string of pointers which mark a path through the data structure. The pointers are followed to find all the elements bound together by one association. If the last pointer points back to the first element in the association, then the structure forms a closed ring of associations and the structure is called a ring structure.

To illustrate the use of data structures, more especially ring structures, and why they are useful (necessary?) consider Figure 2.5. To display the triangle, a ring from its block is traced through to the blocks of the composing lines. From the line blocks, rings to the blocks are followed to find actual coordinate information. Using this coordinate information the display image for the lines forming the triangle can be composed. The complexity of this structure is necessary to facilitate manipulating items in the data structure. In performing some editing function on an item, all the links or ties must be appropriately updated. The deletion of an item, say P_2 means the deleting of L_1 and L_2 for they are no

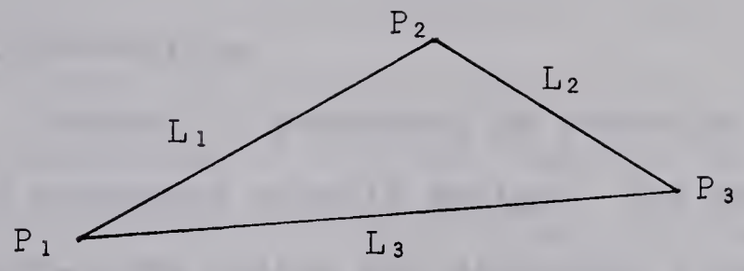
longer defined, and furthermore since L_1 and L_2 are deleted, the triangle is undefined, and it too must be eliminated. This is an example of the type of linking which is required in a data structure to facilitate effective editing and manipulation.

The intent, at this time, is not to discuss data structures further; the concepts and foundations of data structures have received considerable attention from various authors (Ross 1963, Sutherland 1963, Gray 1967, Roberts 1964). In later chapters while discussing specific application programs in circuit design the nature of the data structure used will be illustrated.

2.6 Conclusion

There has been considerable research done in investigating the possibilities of the computer as an aid to the design engineer. As mentioned in Section 2.2.1, there are two problems in circuit design: synthesis and analysis. The bulk of research has been pursued in the use of computers in the analysis portion of design using batch processing techniques. Although objections to this type of analysis exist, this research nevertheless has demonstrated the usefulness of the computer in this work. However, this has contributed very little towards computer-aided synthesis. Until the advent of on-line design the awkwardness of man-machine interaction inhibited the effectiveness of computer-

(a) triangle



(b) data structure

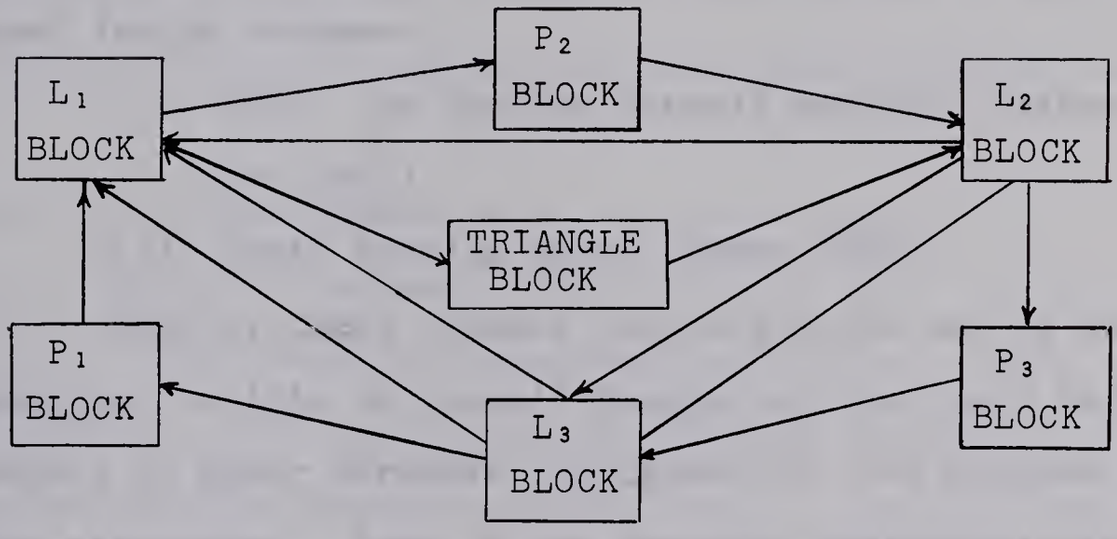


Figure 2.5 Triangle and its data structure

aided synthesis. Research now is most active in investigating the on-line facets of design especially using graphical displays. Some of the objections to computer-aided design using batch processing have been overcome with on-line systems and now the field is extending itself into computer-aided synthesis of electronic circuits. The design engineer is trending towards the "on-line" utility and the language he is speaking is graphic.

CHAPTER III

TWO DESIGN SYSTEMS

3.1 Introduction

Chapter II presented an overview of the state of the art of automated circuit design. The development of computer-aided design was discussed briefly from batch processing to the current trends in on-line design. In this chapter will be given a detailed description of two computer-aided design systems:

(1) OLCA: An On-Line Circuit Analysis System (So 1966)

(2) Logic Drawing Board (Cross 1967)

Both of these systems illustrate the use of an on-line graphical utility in circuit design but they each differ greatly in their hardware configuration, and program and data structures. Some of the dominant characteristics of each system will be described and a comparative analysis of the two design projects will be given. Considerable attention will be devoted to the synthesis part of the systems and there will be little discussion on the analysis programs of the respective systems.

3.2 OLCA: An On-Line Circuit Analysis System

3.2.1 Introduction

OLCA, developed by Dr. H.C. So of Bell Telephone Laboratories, is an experimental software system for on-line

analysis of linear networks. Inputs to the system, such as circuit diagrams, are given via the light pen and the teletypewriter of a remote graphical console. Due to objections to the state of computer-aided circuit design using batch processing (Section 2.2.3), OLCA was built to investigate the effectiveness of on-line computing as a tool for the design engineer who is not necessarily a specialist in computing. The objective of OLCA was to create a system in which the designer could specify his problem naturally and obtain his results quickly and in a form convenient to him.

3.2.2 System Organization

3.2.2.1 Hardware

The operating environment of OLCA is shown in Figure 2.2. It consists of the GRAPHIC 1 Console System (Ninke-1 1965), which is satellite to a central computing facility. The central computer is an IBM 7094 with 32K (36 bit words) of core. The console system can be divided into two units: a control computer, which provides the local console computing power; and the console input devices, which include a display scope and an associated core buffer for storing display material. The control computer at the console is a DEC PDP-5 computer with 4K of 12 bit word, 6 micro-second core. The instruction set allows addition, subtraction, indexing, direct and subroutine branching, accumulation testing and manipulating, but it does not

provide for hardware multiply or divide.

The display scope is a modified DEC Type 340 Precision Incremental Display which allows point plotting on a 1024 x 1024 raster over a 9 $^{3}/_{8}$ inch square scope face. The display material memory is an Ampex RVQ core unit with 4096 36-bit words and a 5 micro-second cycle time.

The main input device for the console system is a DEC Type 370 Light Pen. Additional input devices include Teletype Model 33 ASR, a function keyboard consisting of 32 buttons, and a DEC Type 415A card reader.

3.2.2.2 Software

All problem tasks, except the actual analysis of the circuit are assigned to the console system. The software system of OLCA consists of three parts:

Program A - A display-controlling program written for the PDP 5 computer. This program, written in GRIN 1*, is responsible for both application program specification and result displays.

Program B - A main program written for the IBM 7094. This program, written in GRIN 94*, performs data conversion, command execution, and output direction.

Program C - A circuit analysis program (So-1 1966), written in FAP and FORTRAN for the IBM 7094 to perform frequency analysis of networks.

---

* GRIN 1 and GRIN 94 are graphical input-output languages developed at Bell Telephone Laboratories, Inc.

The last two programs (B and C) are stored on a disk file for the central computer and are automatically accessed whenever needed.

3.2.3 Design Technique

A designer begins a session by loading Program A of OLCA into the satellite computer and the console. An initial display appears on the CRT similar to Figure 3.1. Using the console command language (light buttons) on the top of the screen and the circuit component symbols on the bottom of the screen the designer can construct his circuit with the assistance of the light pen. A part of a typical design session is illustrated in Figures 3.2 - 3.7. Beginning in

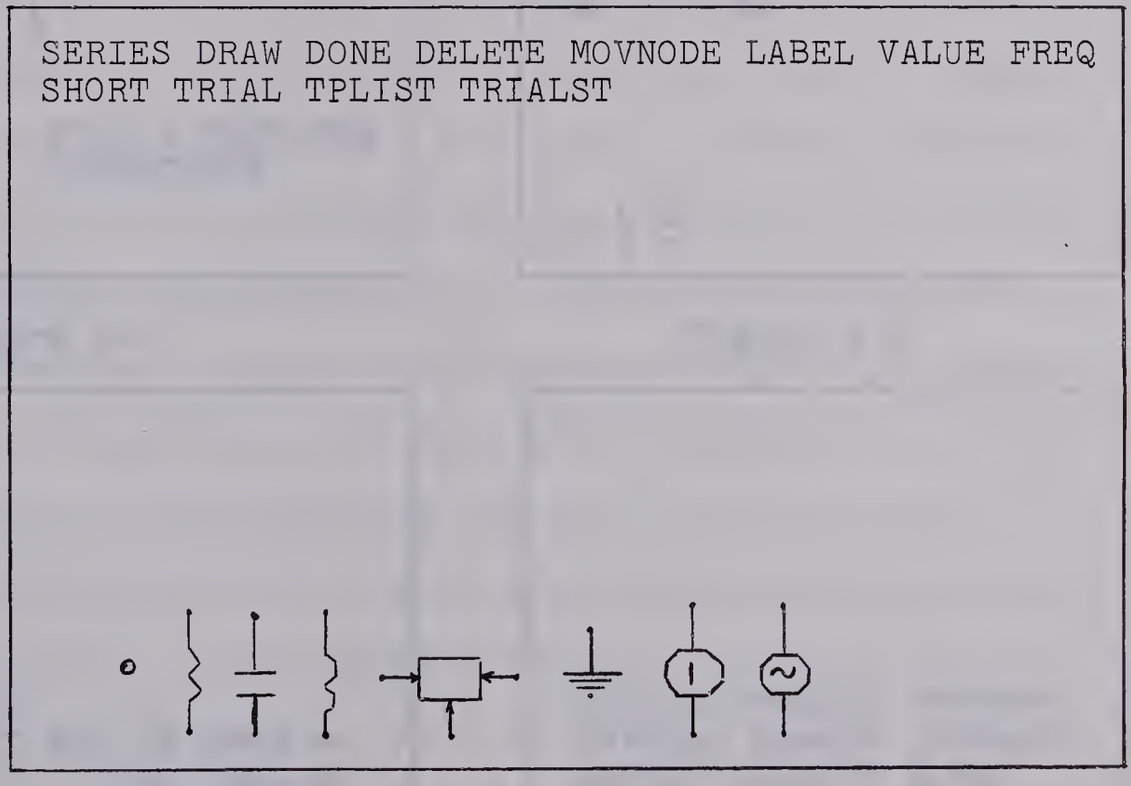
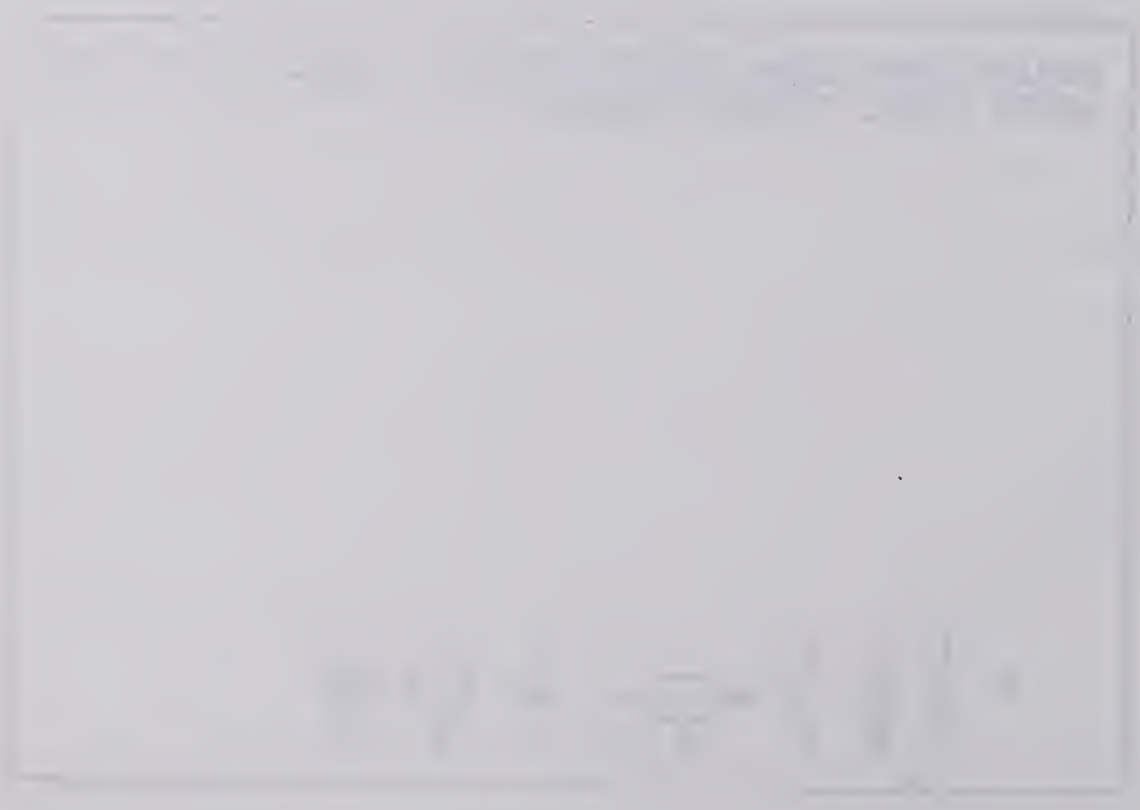


Figure 3.1 OLCA: Initial CRT display



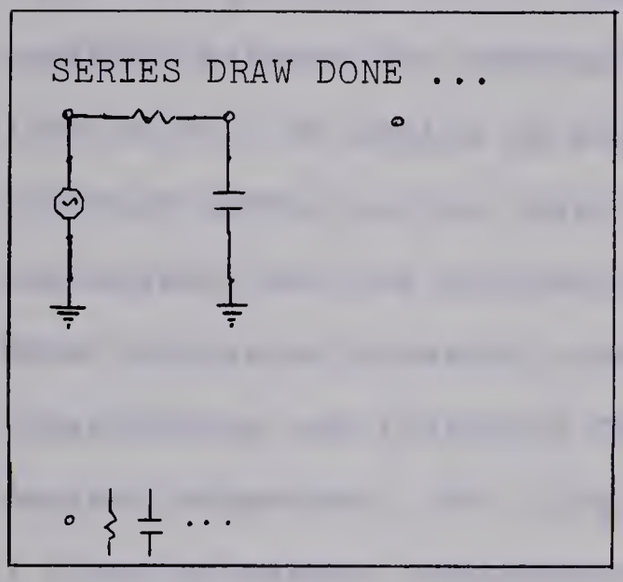


Figure 3.2

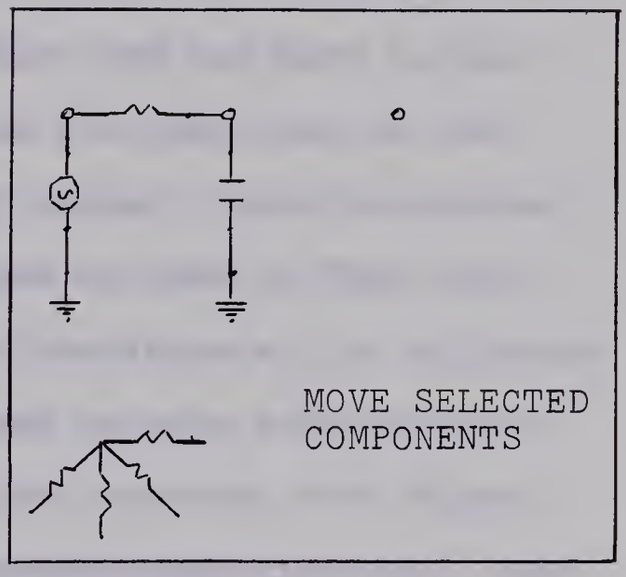


Figure 3.3

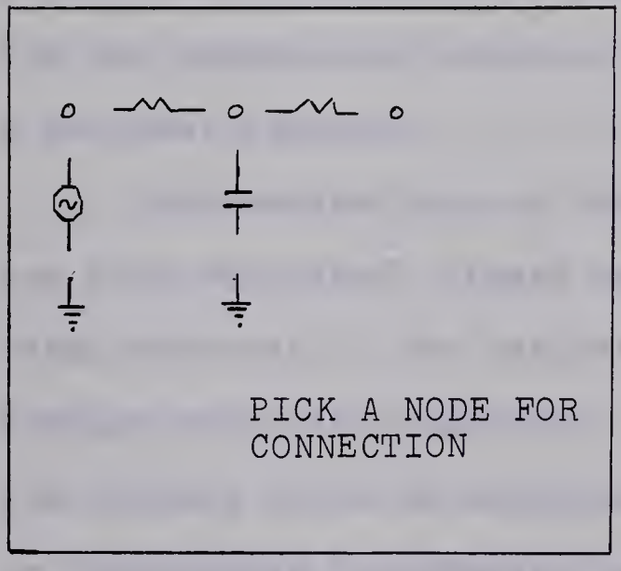


Figure 3.4

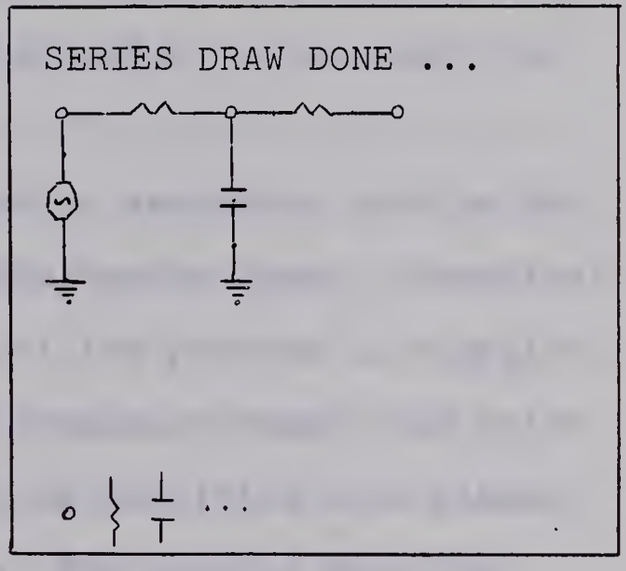


Figure 3.5

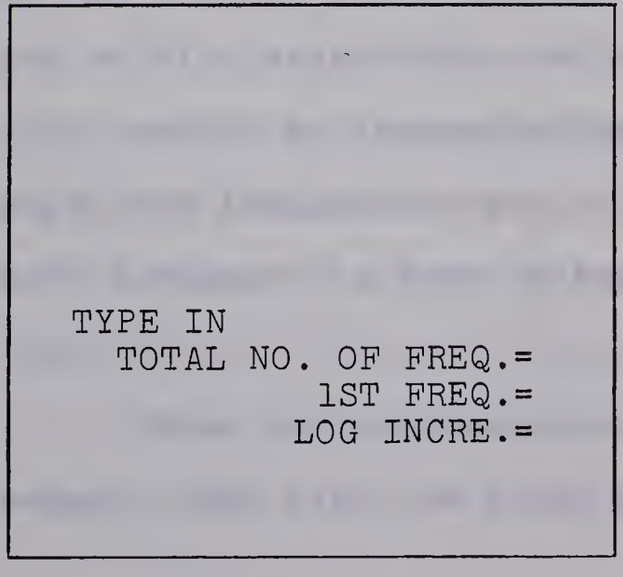


Figure 3.6

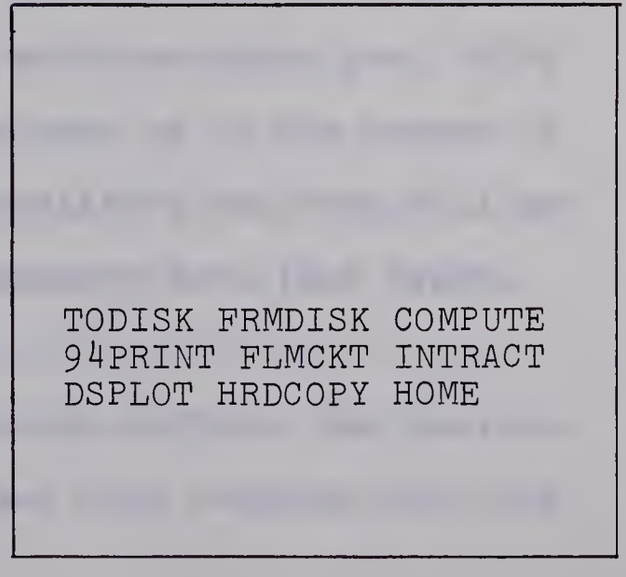


Figure 3.7

Figure 3.2, suppose the designer wishes to connect a resistor between the unattached node and the node to the left of it. He begins by pointing the light pen at the resistor symbol in the lower left corner. OLCA recognizes the request and the display changes to that of Fig. 3.3. Here the system presents a set of resistors all at different orientations and instructs the user to move and place his desired component. The display then responds with Figure 3.4 and a request for node connection identification. When connection information is supplied, Figure 3.5 appears with the new connections drawn in. The system is now ready for a new user request.

By combinations of interaction sequences such as the one just described, linear networks may be drawn. Numerical data pertinent to the definition of the problem is supplied through the teletypewriter. The frequency range over which the circuit is to be analyzed may be specified on a linear or logarithmic incremental basis. For example when the designer wishes to define the frequency range of his analysis he will select the word FREQ with the light pen. OLCA will respond by interrogating the user as to the manner in which the frequencies are to be specified and then will end with a request to type in the necessary data (see Figure 3.6).

When the problem is completely defined, the designer selects DONE with the light pen and OLCA responds with the

options available to the user (Figure 3.7). These options include: (1) store problem on disk; (2) retrieve problem from disk; (3) analyze circuit and store results on disk; (4) analyze circuit and print results; (5) record circuit on microfilm; (6) initiate interaction with the main computer; (7) analyze circuit and plot results at console; (8) analyze and produce rapid hardcopy plots of results; (9) return to main display frame.

This illustrates the design process which can be accomplished using OLCA. Some of the prominent features of the design technique are as follows:

1. All user requests are performed by selecting the light buttons. No function keyboard is used and the teletype-writer is used only to input problem data requested by OLCA.
2. If any action required of the user is not evident, a guiding comment is displayed; therefore, no memorization of complex operating roles is required.
3. To control the actions of the designer, OLCA displays only material (light buttons and/or graphical parts) that the user immediately needs.
4. The console computer provides real-time error checking. That is, if the user does not perform a valid action, then his action is rejected and the program will not proceed until the called-for action is provided.

3.2.4 Data Structure

The data generated by OLCA as a result of user activities is divided into two types: graphical information and non-graphical information. The graphical type of data is stored in blocks of words (segments) each of which contains codes, or a pointer to codes for displaying a graphical item. It is possible, and convenient, to define dummy segments which generate no displays. The non-graphical type of data is stored in blocks called trailers, of which there are three types: node trailers, component trailers, and non-circuit trailers. The trailers are of arbitrary length and may be attached to any segment; thus, any entity is described by its segment, which may be a dummy, and its trailer, if necessary.

Node trailers, as the name suggests, contain the data description for nodes. In a group of nodes which are joined together, one is designated as the master node; the rest of the nodes are slaves. Each node trailer contains the following: a node identifier; a pointer to its label; an indicator of whether it belongs to a shorted group; an indication of whether it is a master or a slave; and a pointer to the master if it is a slave. In the master node trailer, the last is replaced by a pointer to a linked list in which are stored pointers to all slave nodes of the shorted group. Therefore, every node displayed is represented separately in the data structure; it is possible to trace from any node to its master, if any, and on to its fellow

slaves, if any.

Most circuit information is stored in the component trailers. The component trailers contain such information as the component type, and its assigned value if it has variable values. Also the trailers will have pointers to the nodes to which they are attached and also to the circuit label. The non-circuit trailers, which are attached to dummy segments give problem specifications such as output options selected, frequencies desired, and computation requested.

3.3 Logic Drawing Board (LDB)

3.3.1 Introduction

LDB is part of an overall system for computer-aided logic design developed by P. Cross of Cambridge. Cross felt that the on-line CRT and light pen offered a possible approach to logic design. The design system was concerned specifically with the design problems of a tunnel diode project and the aim of the system was to "produce a workable piece of software of particular applicability to this project rather than to achieve great generality".

3.3.2 System Organization

3.3.2.1 Hardware

The Cambridge configuration comprises a DEC PDP 7 computer with 8K of 18 bit word, 1.75 micro-second core and

an extended arithmetic element (EAE). The input/output devices include a paper-tape reader and punch, Westrex KSR-33 teletype, and an 18 button keyboard.

The display scope is a DEC Type 340 display with a light pen. The CRT is a 16 inch tube with 9 $^{3}/_{8}$ inch square raster area.

3.3.2.2 Software

Because of the single processor nature of the hardware system, all program components, data construction, manipulation and analysis is done in the PDP 7. A block diagram of LDB is illustrated in Figure 3.8. An approximation of the relative storage allocation of the program components is shown in Figure 3.9.

3.3.3 Design System

The Logic Drawing Board contains the basic essentials for producing practical diode package drawings. Packages can be created, deleted, or added to existing packages. Packages can be labelled, attached (wired) or detached to other packages. Details of the drawing can then be dumped onto paper-tape and similarly a drawing can also be read in off paper-tape.

The 'drawing board' for the system is an array of 128 x 128 sites, each of which may support one package. The 'board' is toroidal such that the user can extend his drawing in any direction without running over any boundary.

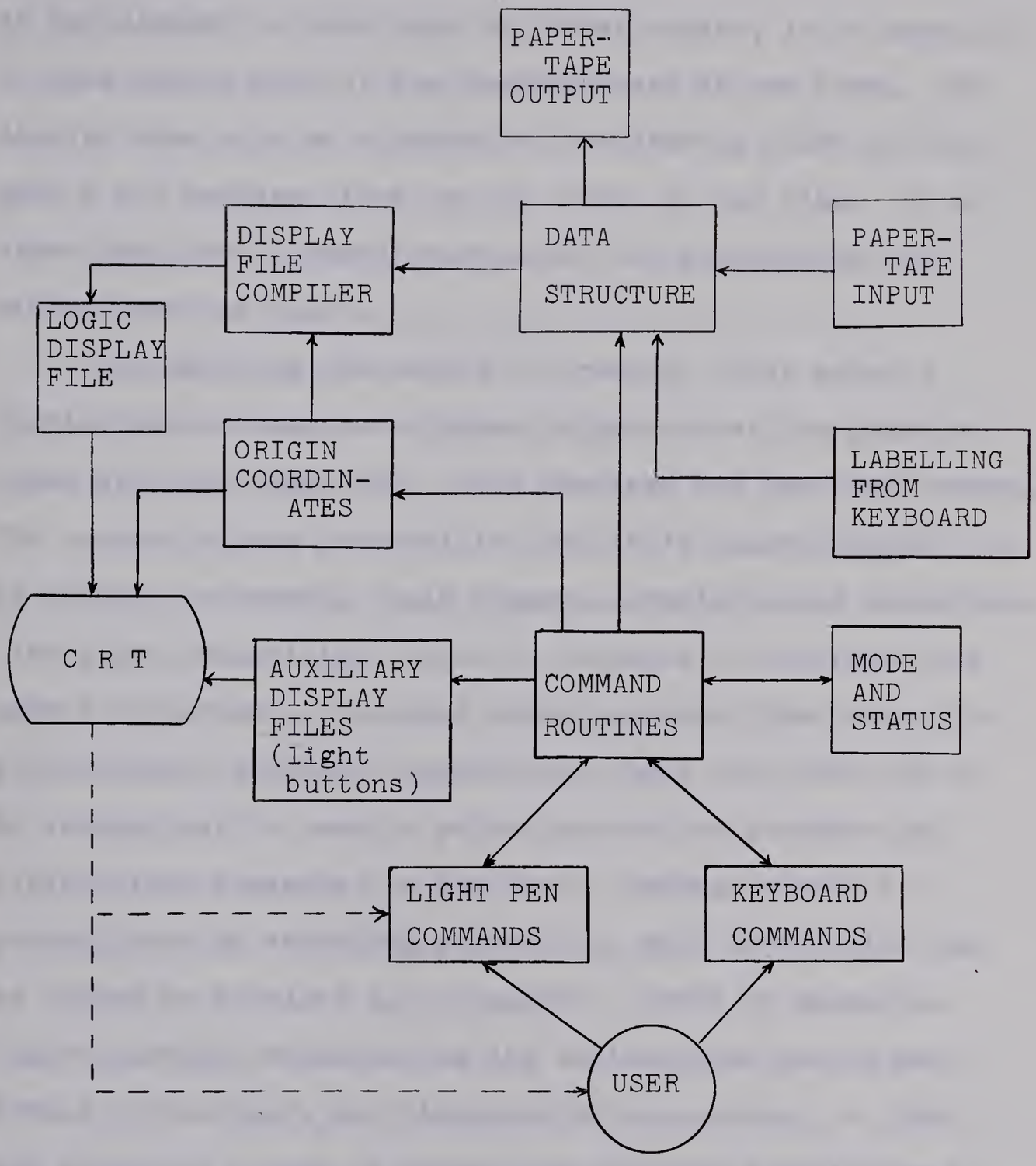


Figure 3.8 LDB Program Structure

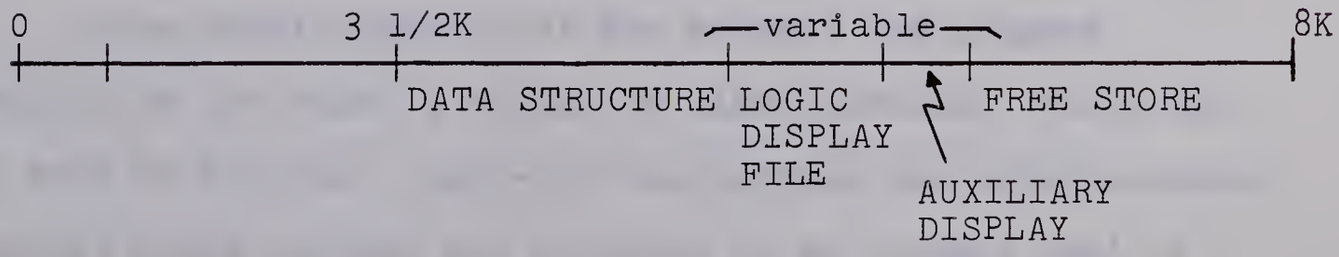


Figure 3.9 LDB Storage Allocation (approximate)

As the display is less than 10 inches square, it is possible to show only a part of the drawing board at one time. The display then acts as a window to the drawing board in which only 7 x 7 package sites can be 'seen' at one time. It is clear then that commands must exist for positioning the window over the board.

In choosing the method of drawing, Cross makes a distinction between two classes of procedures for graphical input with the light pen: Grid Drawing; and Freehand Drawing. The latter is more powerful in that it is unconstrained, but to produce an orderly logic diagram, sophisticated algorithms (character recognition) would be required to interpret the user's intentions. To avoid these problems Cross chose the grid-drawing technique, whereby the light pen never has to be tracked but is used to select one out of a number of alternatives presented to the user. Package wiring is accomplished by selecting points in a grid array which can be joined by straight line segments. Words or mnemonics (light buttons) representing the actions that can be performed by the user, are displayed on the screen. A light pen selection of one of these light buttons constitutes a command to the system to perform the requested action.

Upon initialization of the system, the program displays on the right a column of light buttons. There are two sets of buttons: mode-setting buttons and origin-control buttons; these buttons are referred to as 'normal set' of

light buttons. At this point the function of these buttons will be discussed briefly and then a more detailed description of how they are used will be given.

Mode-setting buttons:

AD: 'Admire Drawing' (The system is initialized in this mode.) Disable packages and wires and turn off auxiliary pictures.

CP: 'Create Package' Disable packages and wires and display the array of 7 x 7 location crosses and prepare for a pen hit to indicate location of new package.

DP: 'Delete Package' Enable packages and prepare for pen hit on a package.

LP: 'Label Package' Enable packages and prepare for pen hit on a package to be labelled.

BW: 'Begin Wire' Enable packages and prepare for a pen hit on a package where wire is to begin.

EW: 'Edit Wire' Enable wires and prepare for pen hit on wire.

RD: 'Read' Input coded drawing from paper-tape and add to current drawing; on completion reset AD mode.

PN: 'Punch' Output current drawing on paper-tape; on completion reset AD mode.

Origin-control buttons:

↑ : Move logic display (drawing board) one step up.

← : Move logic display one step left.

→ : Move logic display one step right.

↓ : Move logic display one step down.

In selecting the above buttons a ring is displayed around the mode button most recently selected and also around the selected origin-control button. It is clear that in selecting the mode-setting buttons the system does not perform any action which is visible to the user. These buttons simply prepare the system for a special set of actions associated with the light button.

Consider the actions available under the various modes:

CP Mode: Only sites not already occupied can be selected. Once a valid site has been hit the site array vanishes and a dummy package is placed on the site. At this time, a set of symbols representing all the package types (AND gate, OR gate, etc.) together with a cross replaces the column of normal buttons. The user can then select a package which replaces the dummy, or if he selects the cross then the choice of site is cancelled.

DP Mode: The selected package together with its input wires, and label if any, disappears. The normal buttons are replaced by a tick and a cross and the other packages are disabled. The user then selects the cross to restore the deletion (avoids inadvertent deletion) or selects the tick which confirms his initial action and purges the package from the system.

LP Mode: The packages are disabled and the light buttons disappear. The label on the selected package is deleted and the system waits for the user to type in the desired label.

BW and EW (Wiring) Modes: Wires are constructed by concatenating straight-line segments; they are considered to begin on the input side of a package and are drawn backwards from this point until terminated. While the wiring is in progress a special set of light buttons for wiring replaces the normal light buttons:

AD: 'Reset AD Mode': Used to escape from the wiring process.

FS: 'Fix Segment': Extends the wire to the last pin seen by the light pen. The program checks whether there is a package on the site, and if so, considers the wire complete.

DS: 'Delete Segment': Delete the last segment of the current wire along with any attached label.

JW: 'Join Wire': Join the current wire to another wire passing through the same pin.

LW: 'Label Wire': Any current labels are deleted, the light buttons are erased and the program waits for the user to type in a label.

3.3.4 Data Structure

In developing a data structure which would represent

the whole drawing, Cross decided that because of the simple nature of logic diagrams (relative to electronic circuit and network diagrams) the full power of a multidimensional list or ring-structure was not needed. Nevertheless, the principle followed was to maintain a data structure independently of any display files and, making no concessions to display coding, to recompile a display file after every alteration to the data structure.

All essential information for regenerating the display file is recorded as data items associated with each package. A package 'owns' all of its input wires, labels, and input labels. A package therefore, does not know to what package its output wire is connected. At the time when the circuit is analyzed, each wire is traced to obtain the coordinates of its far end, and the list of packages is searched for the one with the same coordinates.

Sorting out a wire-join involves tracing all wires to discover which goes through the particular pin where the join occurs.

The outline of a package data structure is illustrated in Figure 3.10. Each package together with its associated wires and labels is represented by a variable-length record in a contiguous piece of store. The package records are also placed contiguously in core. This facilitates efficient use of storage but, however, a considerable amount of storage reshuffling is required every time an alteration is

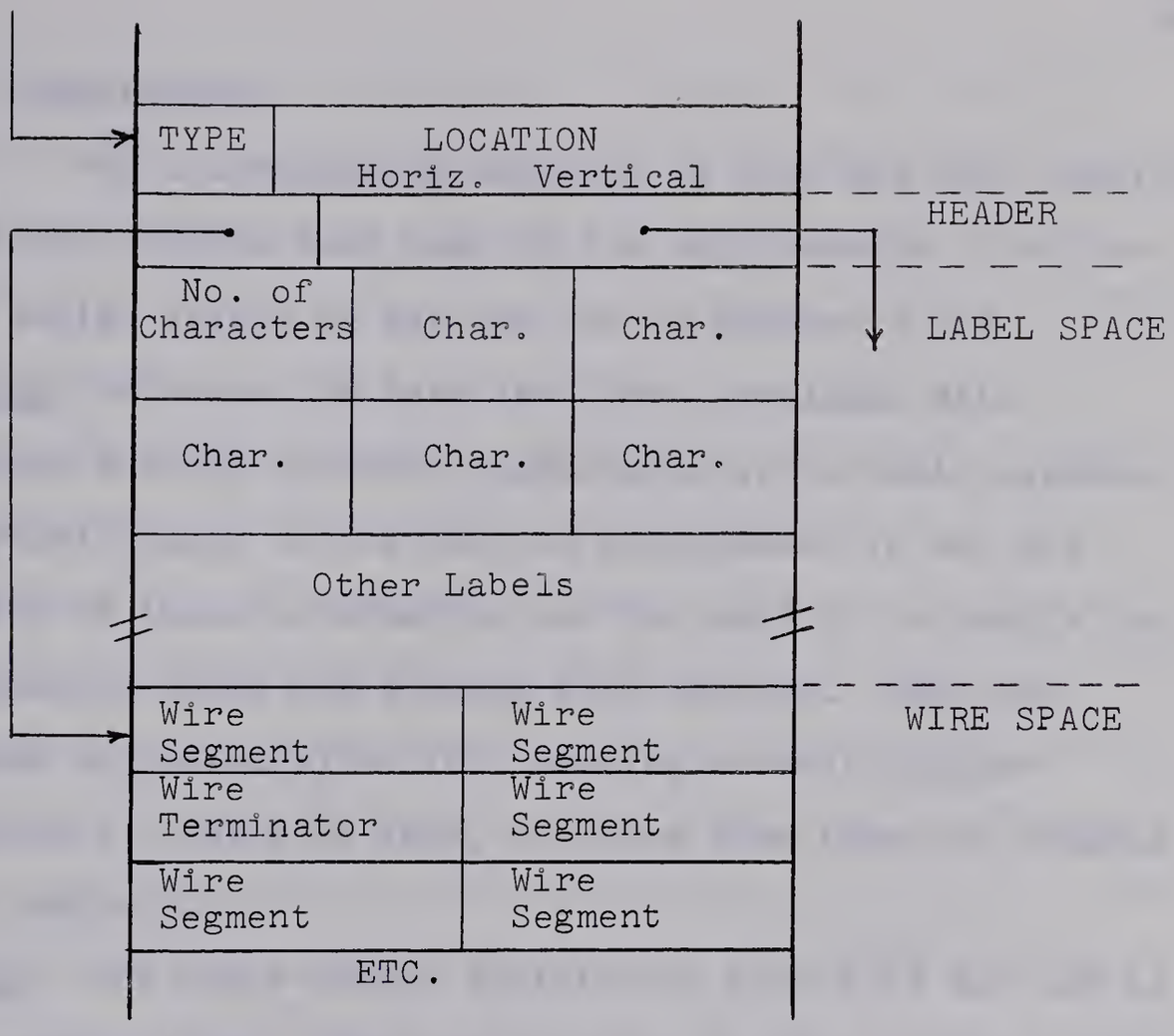


Figure 3.10 LDB Data Structure

made to the logic diagram.

The header of the package data structure contains entries giving the package type and absolute location. Also it contains two relative pointers: one to the first wire segment and a second pointer which points to the first word after the end of the current package entry; in general this is to the next package. The first two words are sufficient to establish a package in the system if there are no labels or wires. Generally the two word header is followed by one or more label entries and then one or more wire segments.

3.4 Comparisons

For a comparative analysis of OLCA and LDB, consider how these systems meet some of the requirements of an on-line design system as was set out in Section 2.3.3.

Editing: OLCA and LDB have each been developed with inherent editing features characteristic in their systems. The significance of the editing requirement is not the ability to input information but the ability to modify the information which has already been entered. Both the systems discussed allow for creating circuit diagram components, adding to them, deleting from them, or purging them completely.

Output: The major output facility of both OLCA and LDB is the graphical CRT. LDB provides for display of the circuit schematics while the more sophisticated OLCA provides for the display of frequency analysis graphs as well as circuit schematics. Hard copies of OLCA's output can also be produced using microfilm records or digital plots. LDB does not provide a hard copy service at its initial development, although this facility is projected for the system.

Definitional: This requirement is not characteristic of either of the two systems studied. Both OLCA and LDB allow the user to perform a predefined set or sequence of operations and there is no facility to allow the user to expand this set. H. So, in analyzing his system has recognized the lack of this facility and he suggests that a definitional feature

would be extremely desirable. Cross, in developing his system admitted at the outset a certain specialization to a design project and so because his system lacks complete generality, he probably did not require an adaptive system.

Informational: As was illustrated in Figures 3.2 - 3.7, OLCA does provide selected information in assisting the user in his design process. At certain points in the program, OLCA indicates to the user what is required of him or it offers a set of alternatives from which he is to choose. LDB provides a more limited informational service to the user than does OLCA. LDB offers little direction to the designer in what action is required of him but it does display some pointers which indicate the current state of his work. For example LDB displays a ring around the mode of operation the system is in and also during the wiring process the program displays a marker on the last pin seen by the light pen.

Diagnostics: In design systems of the type described, the design process usually consists of selecting actions or sequences of actions which the program has offered to the user. Systems of this nature can provide a diagnostic service in two ways: error prevention and error checking. In the first technique the system only offers actions for selection by the user which at that point, in the design process, are valid console commands. For example, suppose in a design system that the teletypewriter is used only

when the designer is typing in a circuit label and that labelling is done only after the user has selected the light button LABEL with his light pen. It would therefore be reasonable for the keyboard to be enabled only after the light button LABEL has been selected and that the light button LABEL should only be displayed when circuit labelling is appropriate. An alternate approach (error checking) is to allow the user to formulate invalid console commands, but then have the system check the form of these commands and if they are invalid, then reject them. It is difficult to have a complete diagnostic service using only one of the above techniques, usually a combination of the two will provide the most effective service.

OLCA has incorporated both methods in providing a diagnostic serivce. The local console facility of OLCA provides real time error checking such that invalid commands are rejected. Also OLCA prevents the user from forming some invalid commands; for example, in Figure 3.4 OLCA does not display any light buttons. Obviously, a reference to one of these light buttons at this time would constitute an invalid command and so the system has prevented this type of console command being input.

LDB provides a similar diagnostic service to the user as does OLCA; that is, by erasing light buttons which would not constitute valid console commands. LDB also provides error protection by disabling (from light pen hits) circuit

components which should not be referenced at certain times. In the wiring process the system will only display those pins to which the wire can be extended legally.

In comparing OLCA and LDB it is obvious that OLCA is the much more powerful and sophisticated system of the two. But a direct comparison of the ability and the structure of the two systems is not meaningful without considering the hardware facilities available to each. Drawbacks to each system can have been created by inadequate system structuring or programming or the drawbacks may have been imposed by hardware configuration. Compared to LDB, OLCA has a powerful computing facility at its disposal; as a result, OLCA has not been unduly restricted in its program structuring. H. So has been able to incorporate a compound data structure in his program which facilitates efficient data searching, manipulating, and editing. There is storage overhead in building compound data structures but the tradeoff between core space and decreased execution time is usually worthwhile. It is this matter of program structuring where LDB is severely hampered. Because of the small core size available to LDB and lack of backup storage facilities, LDB had been forced to make storage economy the dominant consideration in building this system. One of the most serious limitations imposed by LDB on the design process is wire clipping: a wire is only displayed if its parent package is on-screen. Therefore since wires 'belong' to

their inputs, only wires which enter packages on the screen will be displayed; wires which are output from a package and leave the viewing area of the display are not shown. If each package contained pointers to all other blocks to which it is associated, that is, connected, then it would be a simple matter to discover all wires which entered and departed from the package. If a string of pointers was incorporated, then searching of packages to sort out wire joins would also not be necessary. Cross's data structure does not achieve the primary aim in building a data structure (Gray 1967) and that is to model the logic circuit to be constructed. In LDB the logic relationships between the packages are not discovered until the point of circuit analysis is reached. To retrieve these logical associations between the packages requires extensive searching of the package lists. Alternatively, if a simple set of pointers, as mentioned above, were incorporated, then the logic relationships between the packages would be inherent in the data structure such that searching would not be required. With an associative data structure, many of the program segments required for package searching could be reduced or eliminated; this then should make the storage demands of this type of structure more reasonable.

OLCA suffers from one serious drawback and that is its lack of the 'windowing' feature as is characteristic of LDB. The circuit diagrams which can be drawn using OLCA

are limited to approximately a 9 x 9 inch page.

3.5 Conclusion

Without a doubt OLCA and LDB have been successful systems within the constraints of their objectives, that is, to investigate the feasibility of using on-line graphical CRT for electronic circuit design. Both systems have demonstrated that the light pen and graphical display can be useful tools of the design engineer. Although OLCA is a more sophisticated system, LDB is perhaps more outstanding because of what has been achieved within the severe limitations imposed on the system through hardware constraints. Clearly if Cross wishes to build up his system to become more powerful and sophisticated then he would have to expand his computer facility. It would appear that LDB was designed after its computer facility was constructed. Therefore it was built to fit into the PDP 5 as it existed. This is unfortunately, a common procedure in designing application systems but it is not desirable. In an ideal situation the whole application should be designed in the following manner: (1) set out the system objectives; (2) estimate the processing and storage requirements; and (3) set out the response time required for various servicing parts of the program. After this has been done, then the total system requirements are known. Around this system should then be

fitted the computer utility which will service all of the system requirements. An overall effective design system will have to be built in the above such that it does not have to suffer the constraint of a system which was probably designed for another purpose.

CHAPTER IV

PROPOSAL FOR A COMPUTER-AIDED LOGIC DESIGN SYSTEM

4.1 Introduction

The foregoing survey of research in computer-aided design, more especially that using on-line graphical displays, has indicated that further work in this area could be useful. In building on-line design systems, problem areas such as console command languages, data structures, and diagnostics have not been examined fully and as a result, completely satisfactory solutions to these problems have not been forthcoming.

The purpose of the present investigation is to further research in on-line graphical design, and to endeavor to become more aware of some of the fundamental difficulties in constructing systems of this nature. It is felt that the best approach to this type of exploration is to attempt to actually build and implement an on-line design system. In this manner the problems can be squarely met and in trying to solve them, a better understanding should be forthcoming.

Upon deciding to build a system, it is necessary to set out some objectives or goals of the project. One approach would have been to attempt to build a very general circuit design system; one which would encompass a wide variety of circuit design problems such as networks, logic circuits, and integrated circuits. Building such a system

would have presented too formidable a task to complete in the time available. A reasonable alternative is, at the outset, to design something relatively simple and small, but still useful. This should take a relatively short period of time and then when this system is complete, it could be used as a basis for expanding into more generalized and sophisticated systems.

It was proposed therefore to build a computer-aided logic design system (CALD) which could be useful for designing logic circuits, typically of the nature found in digital computers. Since the system is relatively small, its value as a sophisticated design tool is limited. However, it is believed that the system could be particularly useful as a teaching aid in an introductory course in circuit design. Although the analysis of logic circuits is much simpler than that of networks, etc. the graphical requirements, problems, and difficulties are approximately the same. Therefore, since the primary interest is in the problems of graphical communication in on-line circuit design, rather than in circuit analysis itself, the specialization of the project should not defeat the purpose of the investigation.

In the last section of the chapter, there is a discussion of some of the problems which were confronted in attempting to implement the CALD proposal. Before formulating a CALD system, it should be useful to clarify some of the concepts of computer logic design.

4.2 Computer Logic Design

An electronic digital computer is constructed of two basic types of components: logic elements and memory elements. The memory elements store or retain information while the logic elements perform logic functions on the information. A logic element is a collection (large or small) of similar devices, each of which is relatively simple (Hellerman 1967). The logic elements, or logic circuits can be broadly classified as combinational or sequential. A combinational circuit is one whose output at any instant is a function only of its inputs at that same instant. A sequential circuit is one whose output at any instant is a function of its present input and also its past history; a sequential circuit includes the property of storage or memory. A sequential circuit is composed often of combinatorial circuits and time delays in a feedback arrangement.

The logic elements are composed of a collection of elements from a functionally complete set of logic blocks. A set of logic blocks is functionally complete if a logic circuit capable of performing any arbitrary logical function can be synthesized from the elements of the set. One such functionally complete set (not unique) consists of AND, OR, and NOT blocks. The output of each of these blocks has a two value nature; these values could be TRUE and FALSE, HIGH and LOW, or 1 and 0. The behavior of each of these blocks is described by the following:

1. · AND - the output of the AND block is 1 if all of its inputs are 1. Otherwise, the output is 0.
2. + OR - the output of the OR block is 1 if one or more of its inputs is 1. Otherwise its output is 0.
3. - NOT - the output of the NOT block is 1 if its input is 0. Otherwise its output is 0.

Assuming then, the availability of these logic blocks, logic design is the interconnecting of logic blocks to synthesize complex logic functions.

4.3 CALD requirements

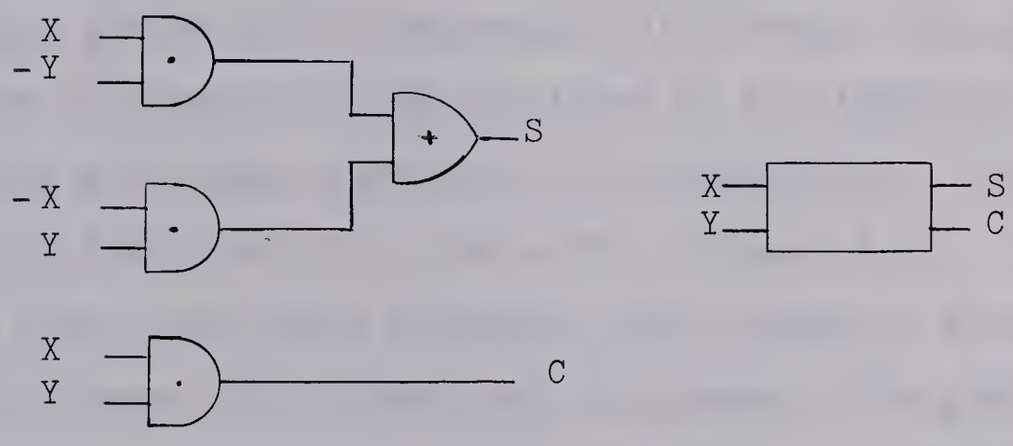
The aim of the CALD system is to provide the engineer with a useful aid to computer logic design. Through use of a graphical display and light pen it should be possible for the designer to construct logic circuits without the user being required to have any programming experience. Some of the desirable features of CALD include the following:

1. A completely light pen directed system in which the user can construct a logic circuit through a series of actions which allow him to create, add, or delete circuit components.
2. Upon completion of the design, the user can specify inputs to the circuit and then CALD will analyze the circuit to produce the proper output.
3. Upon completion (or partial completion) of a circuit construction, the user can request the system to store

its representation of the circuit and then this circuit and other subsequently stored circuits can be retrieved for display and perhaps modification.

4. The user should be able to call for a circuit to become a "black box". For example, suppose an engineer designed a half adder (Figure 4.1(a)) and then tested it until he was satisfied with its performance. He might then wish to design a more sophisticated circuit which required a half adder as one of its components. It would be very convenient if he could then substitute a black box pictorial representation (Figure 4.1(b)) for the one which he had created.

5. The user should be able to obtain a hard copy reproduction of his circuit schematic. The most convenient way to obtain this is through use of a digital plotter.



(a) Circuit (b) Black Box

Figure 4.1 Half Adder

4.4 First Version Objectives

The first implemented version of CALD can be considered a subset of the total CALD system which is described in Sections 4.1 and 4.3. Presently, CALD will allow for the design of combinational circuits with each logic block having a maximum number of two inputs. Features 1, 2, and 5 of Section 4.3 are provided in the present implementation. This version then, as it stands, could be a useful tool in an introductory course in logic design. Simple logic circuits can be built and tested using CALD as it now stands.

One of the most important characteristics of the current version is that it presents an open ended system which can be expanded to more sophisticated levels.

4.5 Hardware/Software Configuration

The implementation of CALD is designed for the graphical display system at the University of Alberta. The details of the configuration are furnished in the Appendix. In general the system consists of two components: the graphical display subsystem and a large CPU. (Figure 4.2). The CPU is a large high speed processor with access to abundant core storage (768K bytes) and disk space. The graphical subsystem consists of a CRT, light pen, alphanumeric/function keyboard, teletypewriter, small processor and a small memory module (4K 12 bit words). The subsystem can operate as a "stand alone" unit but, because of its size,

it is somewhat limited in its uses unless it is operating with the S/360. The small processor executes a program called the "display file" which is contained in the processor's memory. The execution of this program generates the display image on the CRT. While executing the display file, the processor can accept interrupts from the light pen and the keyboard. The processor can service some interrupts, but generally it will simply assemble an interrupt (or a sequence of interrupts) into a message to be transmitted to the S/360 where the message can be processed and the requested action implemented.

Because of the two processor nature of the hardware configuration, the software support for the system is in two parts: one for the graphical subsystem and the other for the IBM 360/67. By reason of the low powered processing capabilities of GRID, it was proposed that the major portion of the CALD processing would be done in the CPU. Apart from containing the display file program, GRID possesses a small supervisor whose functions are the following:

1. accept light pen and keyboard interrupts
2. assemble sequences of interrupts into an interrupt message
3. transmit the interrupt message to the S/360
4. return control to the display file after an interrupt has been processed.

Since the supervisor and display reside and operate in GRID, their coding has been done in the low-level machine language of GRID.

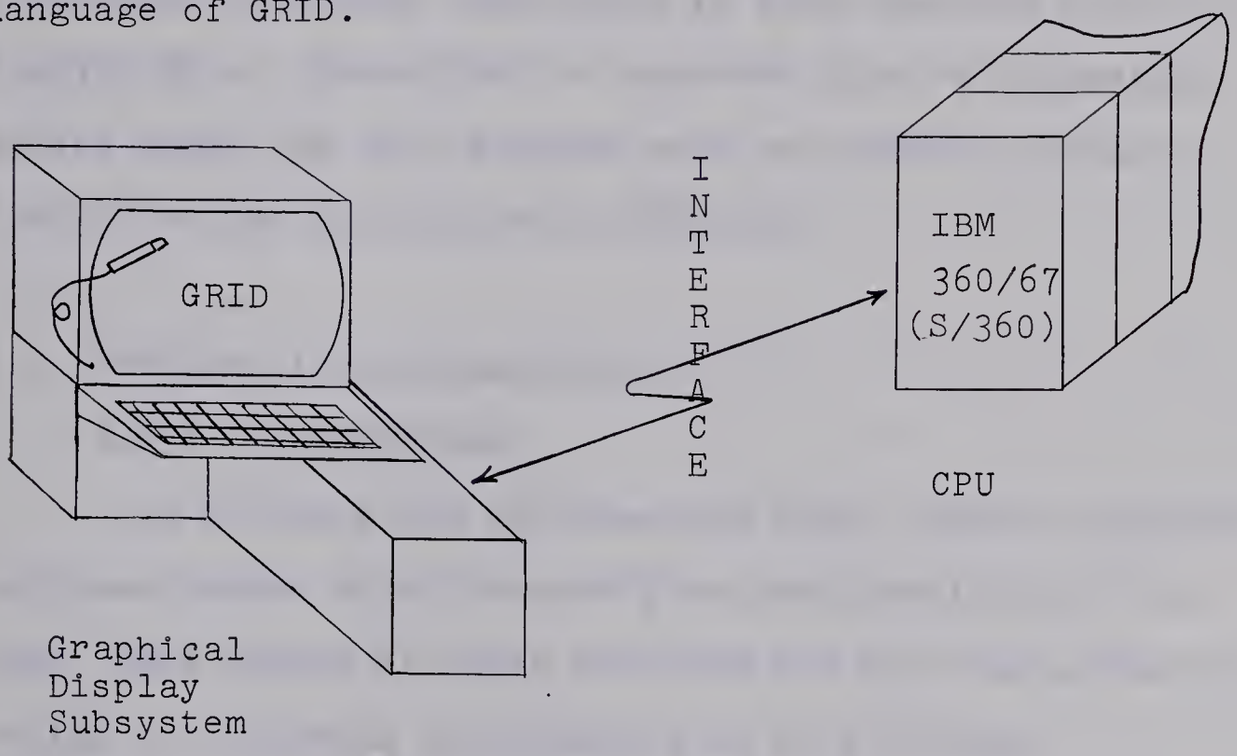


Figure 4.2 Hardware Configuration

The CALD program which is resident in S/360 has been written in FORTRAN IV, a high level language. This program services all of the interrupt messages and then generates modifications for the display file. In order for the CALD program to avoid the generation of low-level GRID coding, the program makes extensive use of a graphical subroutine package, GRIDSUB (Huen 1969). This package contains graphic element generation routines which generate line segments, points, and characters. GRIDSUB provides

facilities for structuring blocks of graphical data, which correspond to picture components. These routines convert the blocks into GRID code which is then inserted into the display file. Therefore, to generate picture components or delete them, the CALD program need only make a series of "calls" to the subroutines in GRIDSUB.

4.6 Problems in Implementation

4.6.1 Introduction

In building and implementing CALD, certain dominant problems ensued which deserve some consideration at this time. The nature of these problems and how they affect the design of CALD will be discussed in this section.

4.6.2 Console Command Language

The console command language is the interface between the graphics subsystem and the designer. With this language the designer communicates with the CALD system, and transmits all of his design requests. In one sense, the CALD system is an interpretive system which can decode and execute a series of commands expressed in a command language. The definition of the language is much at the discretion of the builder of the system. The power, convenience, and simplicity of the design system is thus very dependent on the nature of the console command language.

The character of this language is a reflection of the user requirements and the system configuration. The language

should be powerful, concise, and yet convenient for the user. One of the aims of an on-line graphical design system is to incorporate a system which allows the user to communicate in a language which is natural to his design process. The console language should not reflect or require a prior knowledge by the user of how the system is coded or configured.

The nature of the console command language can cause it to be classified generally into one of two types, according to the lengths of the console command messages. To clarify terms, consider a console command action as any single action with the light pen or keyboard at the graphics console. A console command message is then a string of one or more console command actions which is sent to the S/360. A design action is one or more command messages which is required to complete a single design step.

The first type of command language is characterized by very short console command messages of one or at the most two console command actions. In this instance, several command messages are required to execute a design action. The second type of command language utilizes console command messages composed of several console command actions. Each command message will normally constitute one design action.

Each of these two types of command languages has its own advantages and disadvantages. The main advantage of the

first type of language is that it allows the control program to exercise a greater amount of assistance to, and control of the user's actions. For example, suppose a desired design action was to create an AND gate at the position (x,y) on the screen. The design action is composed of three command messages; these messages are light pen picks of the following elements on the screen respectively: COPY, ⊐, (x,y) coordinate position. After the system accepts the first console command message, COPY, it can perform one of a number of services. It could send back a message which tells the user what he can COPY, it could disable most of the screen such that only the elements which he can COPY are made sensitive to the light pen, or it could only display the items on the screen which the user can COPY. This type of language can provide for a great deal of user guidance and prevent him from executing invalid design actions. The disadvantage of this type of language is that it is expensive in terms of the attention requests which are required of the CPU. Also if the response time of the requests is slow, then the whole design process could be slowed down considerably.

The second type of console command language suffers from the inability to have the system check design action components since normally each design action is equivalent to one console command message. On the other hand it is a more economical language as far as CPU requests are concerned and

if the response time is slow then there would be less time spent waiting for the system responses.

For the CALD system it was proposed to develop a console command language of the second type. This decision was based on the following reasons: (1) the anticipated response time for servicing in the S/360 was in the order of 1 sec. which was felt to be too slow for an environment with multiple transmissions; (2) it was hoped that a console command language could be developed that was simple and convenient, so that every console command action would not have to be checked for correctness.

4.6.3 Data Management

The aim of Computer-Aided Design is to create in the computer a model of the object being designed (Gray 1967). It is required that the CALD system build a model of the logic circuit in the computer as the designer builds the schematic of the circuit on the screen.

It is important to realize that a logic circuit has more than one representation. A logic circuit is a set of logic components which perform a defined function. The circuit can be described by the function it performs or by a pictorial diagram. It is clear that an infinite number of pictorial descriptions exist for a given circuit. The designer builds just one pictorial representation of a circuit on the screen using the CALD system. The functional representation of a logic circuit can be depicted using

Boolean Algebra. For CALD to analyze the circuit, it is this last description of the circuit with which it is concerned. Therefore, some structure must be built up in the computer which is different from the graphic data required to generate the schematic on the display. This structure can contain information related to the generation of the circuit diagram but the pertinent and highly essential purpose of this structure is to model the functional representation of the circuit.

This structure, referred to as the "data structure", must conform to certain criteria if it is to be a useful model. Since this model is representative of the circuit, it must be capable of dynamic growth, and since one may wish to reference any component of the circuit at any time, it must be possible to enter the structure at any point. Apart from this required flexibility of the data structure, there are two main considerations which determine the nature of its design: the amount of storage required and the speed of access to all or any parts of the structure. These two considerations generally conflict and so certain decisions must be made as to which characteristic is most important. Since CALD is an interactive system, speed is a priority item and so this has been the main consideration in developing a data structure.

4.6.4 Diagnostics

From a user's point of view, the purpose of CALD is

to allow him to construct logic circuits and analyze them. Given the specifications for the console command language the task of writing a program which accepts valid console commands is not unduly difficult. However, the problem is made much more arduous by the fact that any design system of a nature similar to CALD not only must be able to accept valid console command messages, but it must also be able to accept invalid command messages and then be able to interpret them as such. The system must respond to an invalid command by displaying a relevant error message; relevant in that the diagnostic will not just say that an invalid command message has been executed, but rather it should give some indication as to the nature of the error.

4.6.5 Circuit Analysis

It can be a complex problem to develop an analysis program which can take any combination of gates and, given the inputs, evaluate the output. To develop an efficient program it is necessary that the analysis program (AP) execute an algorithmic procedure which will systematically analyze the circuit. Consider an example of a simple design of an exclusive OR circuit in Figure 4.3. In the computer there will be a data structure model for this circuit. Latent in this structure is a tree which can be retrieved to appear like Figure 4.4. It is not necessary to actually retrieve the tree for the data structure should provide an adequate representation of the tree structure.

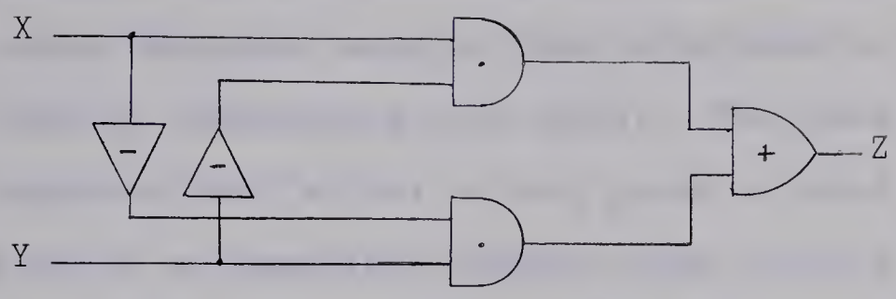


Figure 4.3 Exclusive OR Logic Circuit

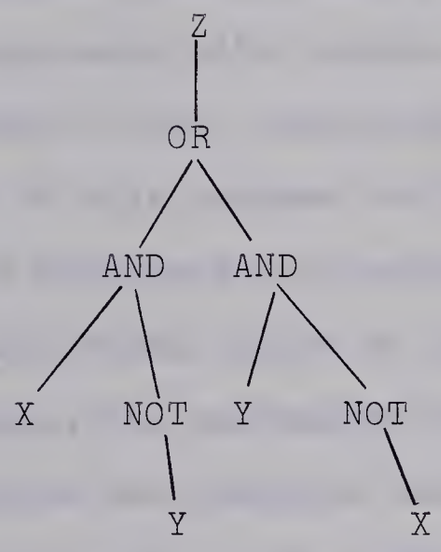


Figure 4.4 Circuit Tree

This tree structure provides a clue towards an approach to the circuit analysis problem. The order in which circuit analysis could be performed corresponds to a left-to-right, bottom-to-top ordering of the tree nodes. One can see an

analogy with the generation of syntax rules by a "top-down" syntax analyzer such as that developed by Cheatham and Sattley (Cheatham et al. 1964). The data structure could be searched with either of two goals in mind: the first is to produce an immediate output value for the circuit, knowing the input values; the second is to map the circuit onto an executable program which can be called to analyze the output.

The first approach constitutes a search down the tree stacking logic components with unknown inputs until a known input value is found. Then, operating in a manner similar to the generation of rule numbers in Cheatham and Sattley's analyzer, AP could progressively evaluate each logic component output until the final output of the circuit is known. For different inputs, the successive evaluations of the circuit would require the complete search to be repeated.

The second approach requires AP to map the circuit onto an executable program. Using almost the same program as in the first technique, AP could generate the Boolean equation which represents the circuit. For example using Iverson's notation (Iverson 1967) the exclusive OR circuit could be mapped onto the following equation

$$Z = (X \wedge (\sim Y)) \vee (Y \wedge (\sim X))$$

From this equation could be generated an executable program which, given the values of X and Y, would output the value of Z. The procedure of creating the program could be fairly lengthy but the execution of the program would be

quite short. For repetitive executions with different input values, the output value could be evaluated by just calling the program generated by AP in the first analysis of the circuit.

In comparing these two techniques it is obvious that in an environment where a circuit was being repeatedly analyzed and then modified, the first type of AP would be more reasonable. There is no sense in going to a lot of work to generate a circuit program if it is to be discarded immediately when the structure of the circuit is changed. However, when the design of the circuit has stabilized, that is, perhaps the user is ready to "black box" his circuit (Section 4.3), this then would be an appropriate time to have the circuit analyzed in a manner which would map the circuit onto an executable program.

It is proposed then, that as the complete CALD system is developed, both types of analysis programs would be valuable. For the present the first type of analysis program has been selected and the procedure described is the one used to evaluate all logic circuits.

CHAPTER V

CALD DESCRIPTION

5.1 Introduction

This chapter presents a detailed description of the version of CALD as it has so far been implemented. Firstly, the design system from the point of view of the user will be discussed; this will be followed by an examination of the program structure of CALD and the data structure which it creates and manipulates.

5.2 Design Technique

The CALD system is just one application program which is operational on the GRID - S/360 display system at the University of Alberta. At the beginning of a design session, CALD must be loaded into the S/360. After being loaded, CALD displays a message notifying the user that the system is loaded, and then initializes the screen to a display similar to Figure 5.1.

On the upper left hand border of the screen is displayed three prototype logic gates: AND, OR, and NOT. Logic gates identical to the prototypes can be created and displayed at various positions on the screen. The gates can be displayed in one of four different orientations as illustrated in Figure 5.2. The gates can be created at any orientation and also, once they are displayed, they can be

PLOT 1 9 SEPTEMBER 196

Figure 5.1 Initial CALD display

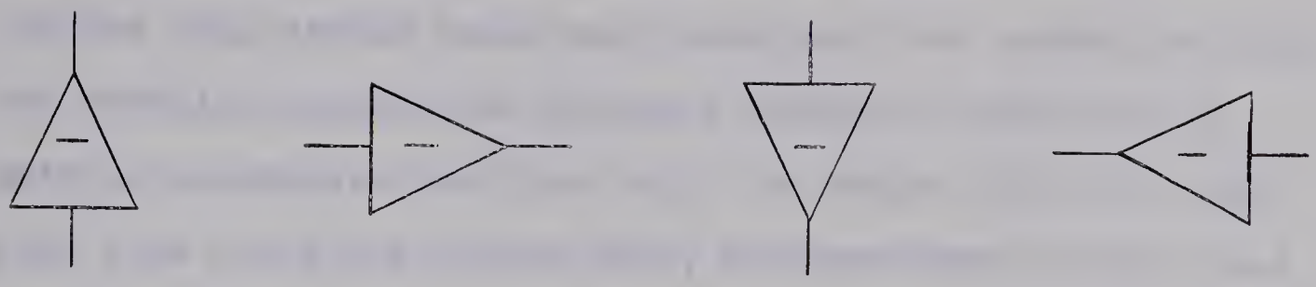
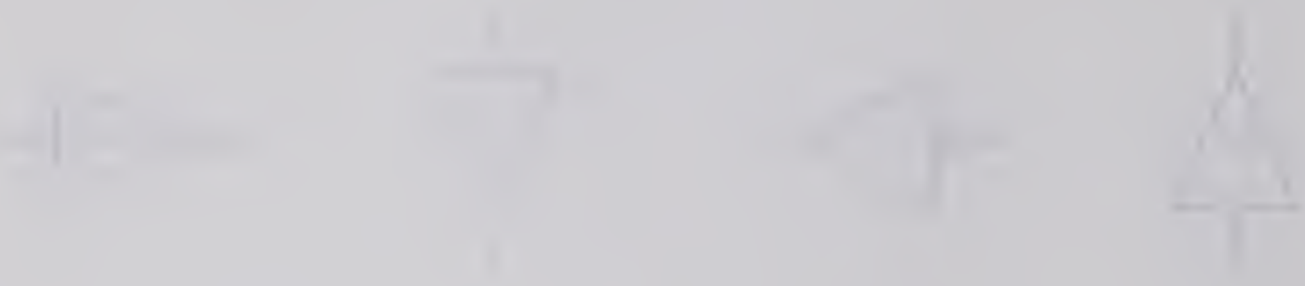


Figure 5.2 Orientations of a gate

rotated to a new orientation if desired.

Below the prototype gates is displayed a symbol resembling an asterisk; this is called a node and is used to denote branch points in wires. A special notation is required to distinguish branch points from wires which just cross over each other. The node can be displayed in the same manner as the logic gates except that it has only one orientation. Although it performs no logic function, it can have one input and any number of outputs.

Following the prototype gates and the node on the screen is a set of command words or "light buttons". These light buttons constitute the set of console command actions which can be performed by the designer. The console command messages or design actions are all sequences of one or more light pen picks of the light buttons, followed by picking one or more of the prototype blocks, screen coordinates, and/or circuit diagram components. All console command messages must commence with the picking of a light button;



this action serves to classify the type of console command for the CALD system which must interpret the command message. For example, suppose the designer wishes to create an OR gate at coordinate position (x,y); he would, with the light pen, pick the light button COPY, the prototype OR gate, and an (x,y) position on the screen. Alternatively, he could pick the light button COPY, the (x,y) coordinate position, and then the prototype OR gate, but he must first start by picking the light button COPY.

A person familiar with the operations of a light pen will realize that it cannot pick a coordinate position from a blank screen; the light pen is a "light sensing" device and so it can only pick up the coordinates of points, lines, symbols etc. already displayed on the screen. The problem then of picking up a coordinate on a blank screen can be overcome in a number of ways such as employing a tracking cross or a screen scan. The technique used in CALD is to display on the screen a grid of dots spaced about half an inch apart. These dots can be likened to logic gate sites or wiring pins in other design systems. This raster of points is activated by depressing key F1 on the function keyboard and it is erased by depressing key F2 on the function keyboard.

Whenever it is necessary to pick up an (x,y) coordinate from the screen this grid is activated and a position can be picked. Because of the use of the grid of points there is

a set of discrete positions available on the screen for placing logic gates and wire segments.

Consider now a brief description of the console commands as annotated by the light buttons.

COPY - permits the copying of a prototype block onto the screen at discrete positions.

ERASE - permits the erasing of logic gates and logic inputs on the screen.

ATTACH - permits the interconnecting or "wiring" of logic gates. The "wire" can extend directly between input leads, output leads, and/or nodes or the wire can be segmented by extending from an I/O lead to one or more coordinate positions and then to a final I/O lead.

DETACH - permits the erasing of wires.

ROTATE - allows an unattached gate to be rotated to any of four orientations. The orientation desired is indicated by pointing to one of four directional arrow heads displayed.

MOVE - allows for a gate already displayed to be moved to a new screen position.

INPUT - permits the applying of a logical value (0 or 1) to an input. A "0" and a "1" are displayed as light buttons and are selected in a manner similar to other light pen selections.

ANALYZE- permits the designer to select an output such that

the circuit will be analyzed and an output value given.

OUTPUT - assuming the circuit has been analyzed this allows the user to interrogate the logic value (output) at any selected lead or node.

E-D - permits enabling and disabling of the light pen for detecting the connecting wires.*

RESTART- all of the current work is deleted and the system is reinitialized.

PLOT - provides for producing a hard copy (on the plotter) of a display file.

FINISH - terminates CALD program.

To illustrate a typical design session consider the following sequence of operator actions with equivalent system responses shown in Figures 5.3 - 5.8. (XY denotes an (x,y) coordinate position on the screen and $ is used as a delimiter between console actions).

Operator Action: With light pen, pick: COPY $  $ XY $
Hit SEND key (SK).

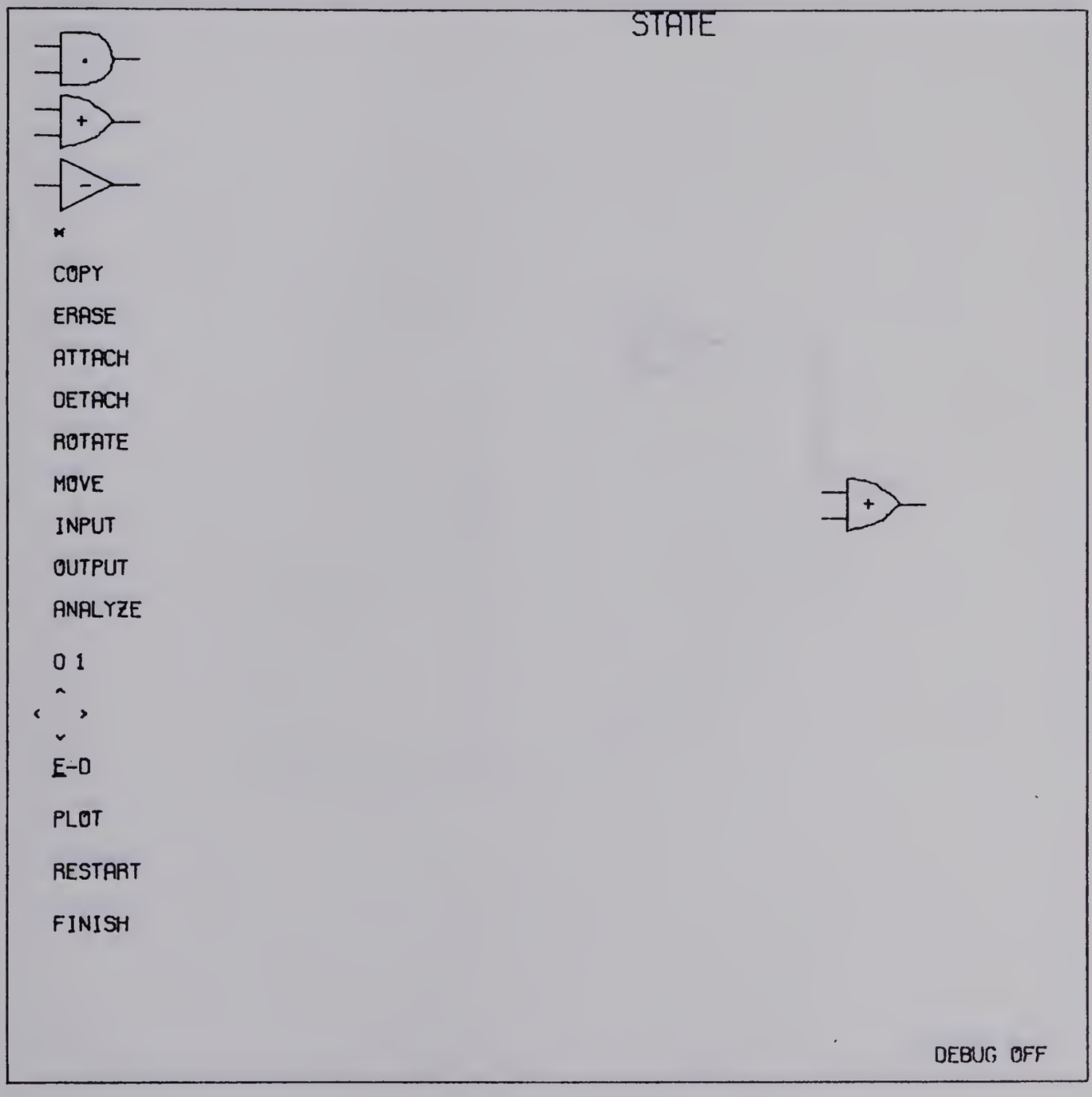
System Action: See Figure 5.3.

Operator Action: With l.p., pick: COPY $  $ XY $ SK.
With l.p., pick: ATTACH $ output of AND gate $ XY $ input of OR gate $ SK.

System Action: See Figure 5.4.

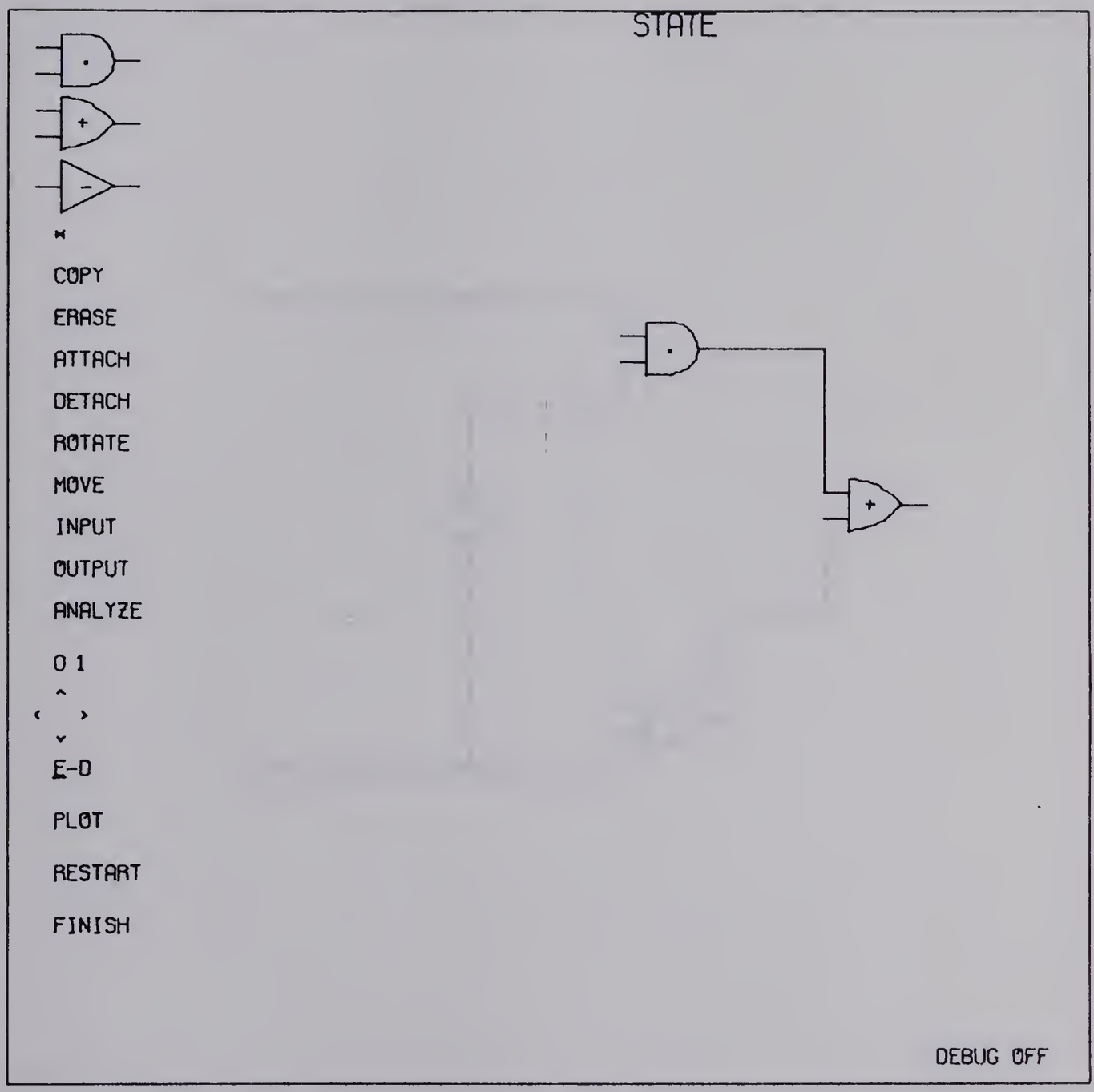
---

*This is necessary to resolve problem of wires intersecting at a node. If there are one or more wires intersecting at a node it is impossible to insure a light pen pick of the node unless the wires are made not detectable.



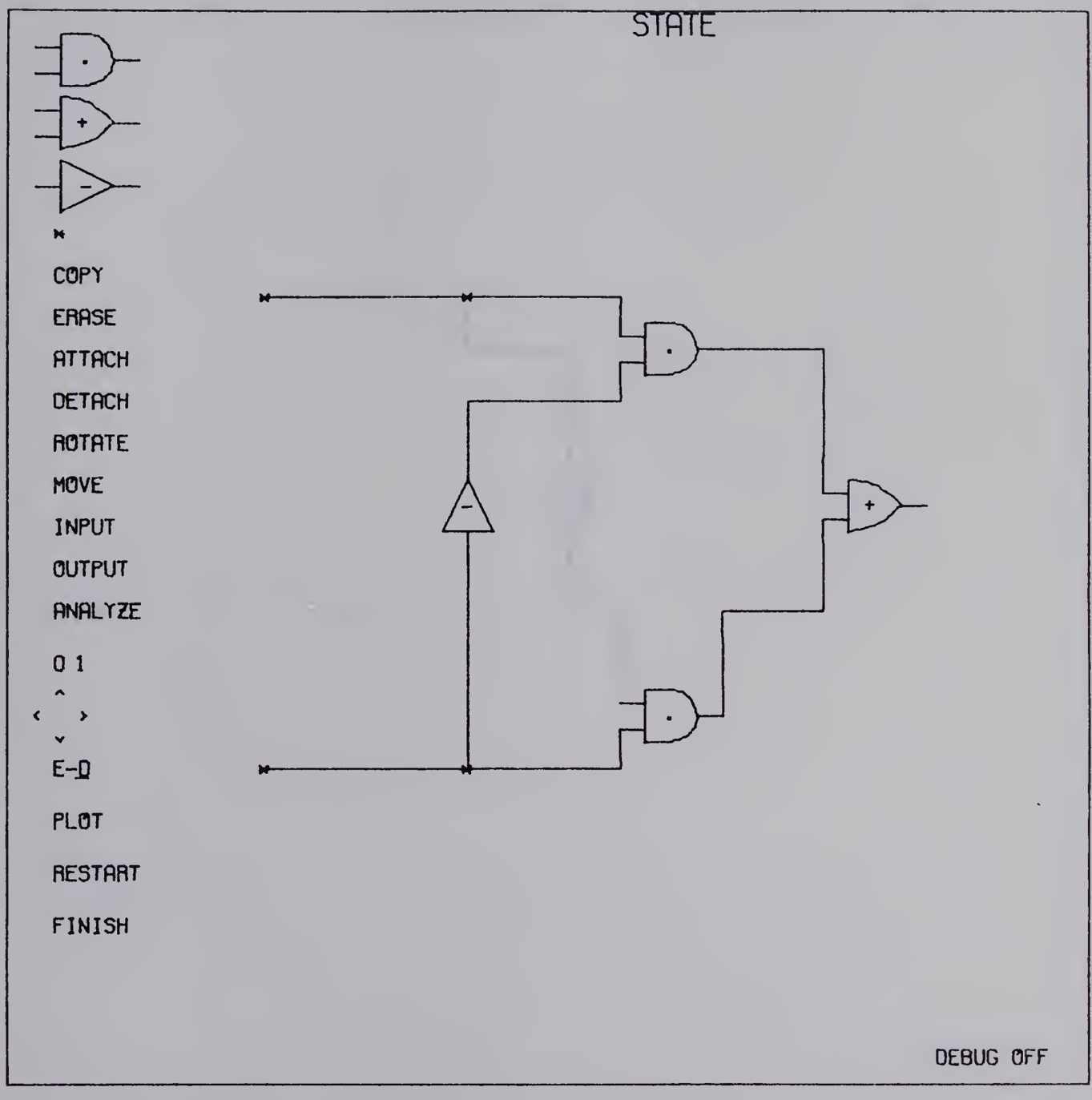
PLOT 2 9 SEPTEMBER 196

Figure 5.3 OR gate



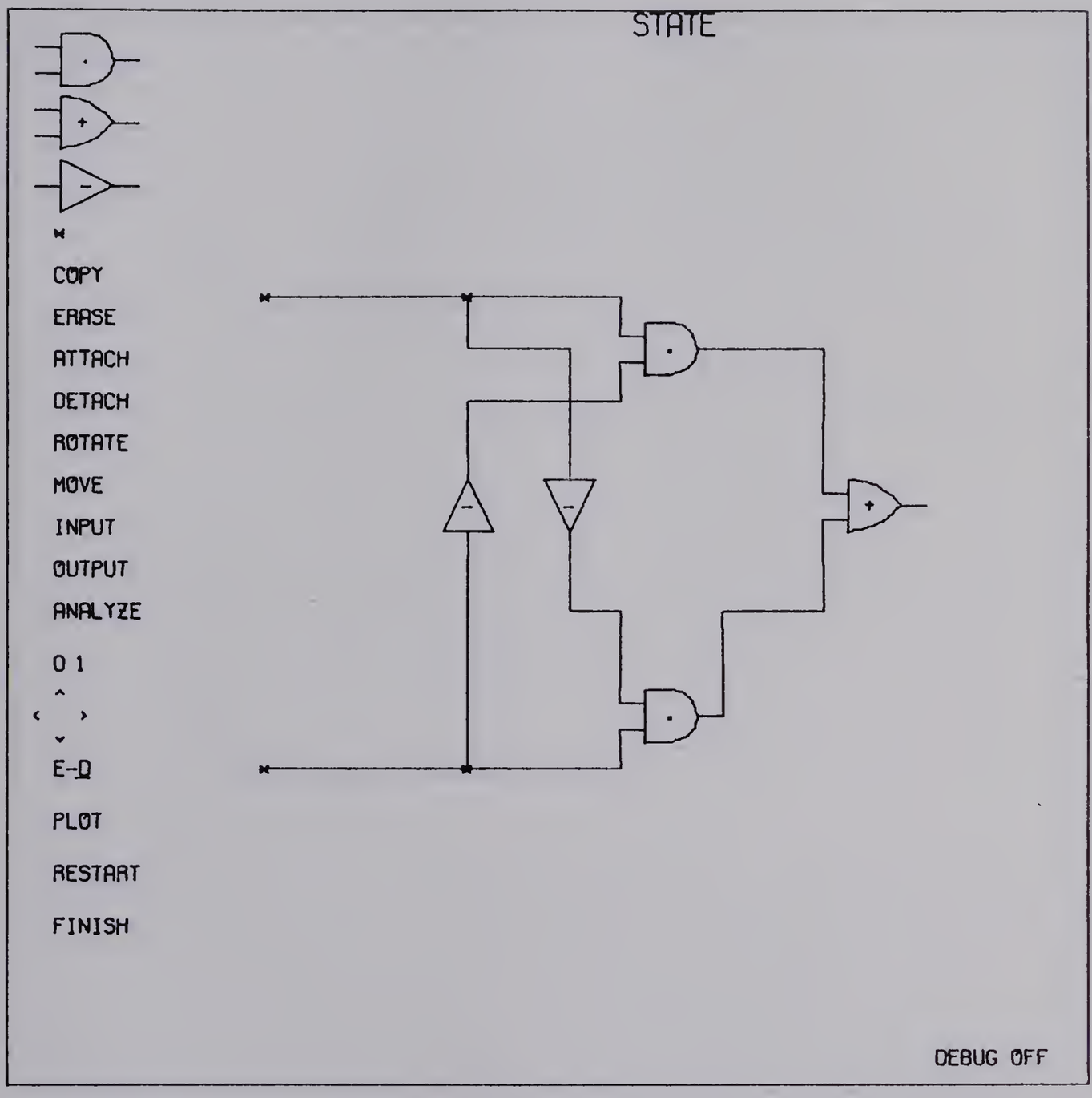
PLOT 3 9 SEPTEMBER 196

Figure 5.4 AND gate attached to OR gate



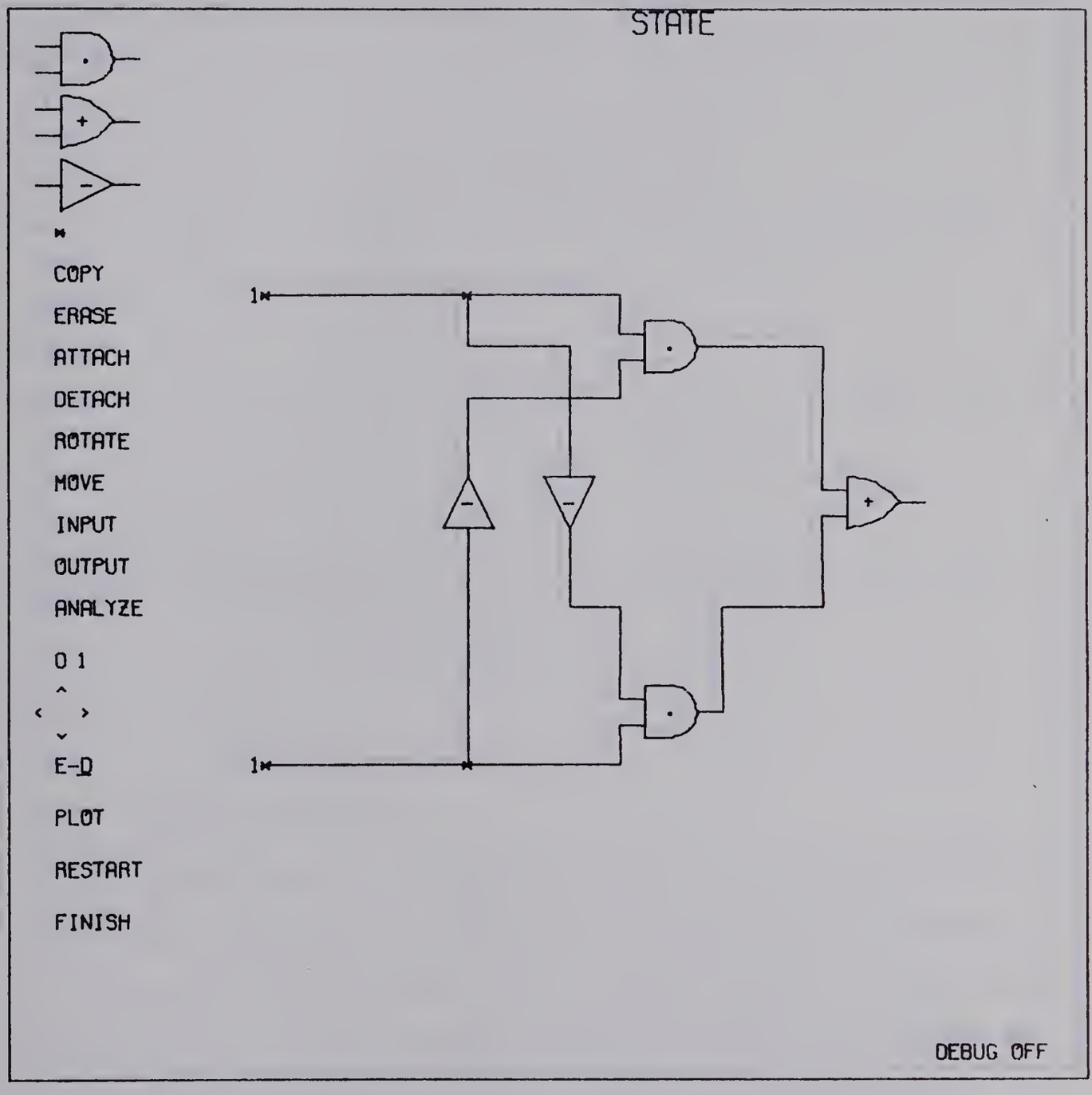
PLOT 4 9 SEPTEMBER 196

Figure 5.5 Incomplete logic circuit



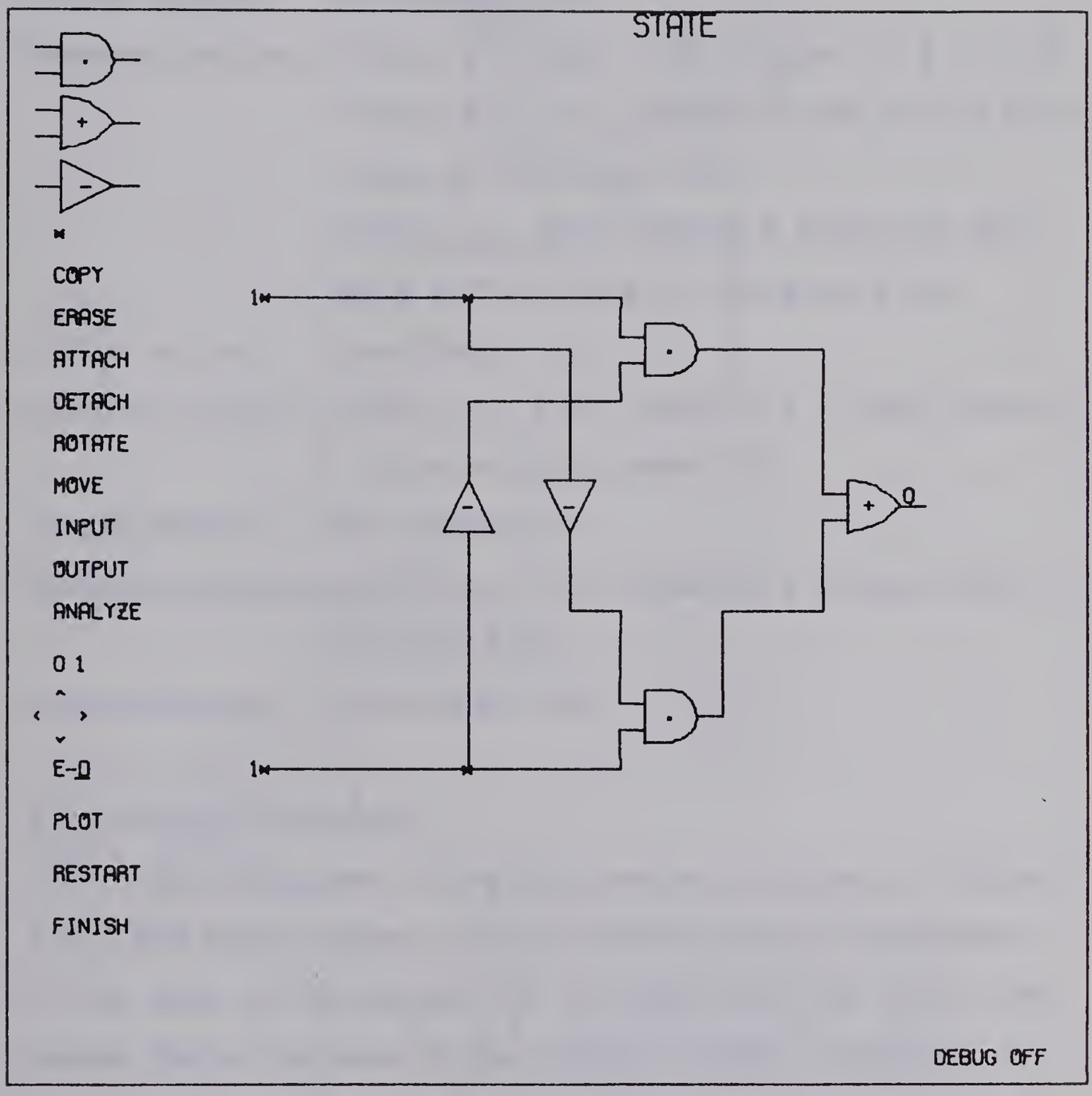
PLOT 5 9 SEPTEMBER 196

Figure 5.6 Exclusive -OR circuit



PLOT 6 9 SEPTEMBER 196

Figure 5.7 Exclusive -OR circuit with inputs

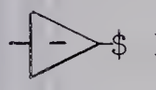


PLOT 7 9 SEPTEMBER 196

Figure 5.8 Analyzed circuit

Operator Action: A series of COPY commands followed by a number of ATTACH commands.

System Action: See Figure 5.5.

Operator Action: With l.p., pick: COPY $  $ XY $ ⌄ $ SK.
With l.p., pick: ATTACH $ node $ XY $ XY $ input of NOT gate $ SK.
With l.p., pick: ATTACH $ output of NOT gate $ XY $ input of AND gate $ SK.

System Action: See Figure 5.6.

Operator Action: With l.p., pick: INPUT $ 1 $ input node $ 1 $ other input node $ SK.

System Action: See Figure 5.7.

Operator Action: With l.p., pick: ANALYZE $ output lead of OR gate $ SK.

System Action: See Figure 5.8.

5.3 Program Structure

The structure of the CALD system is shown in Figure 5.9. The exact nature of this structuring is transparent to the user of the system; he is aware only that he is executing design actions at the console. GRID contains a supervisor which accepts and assembles interrupts into a message, and a display file, which contains the necessary coding to display a picture. The console command message is sent to S/360 where the main CALD system receives it. First, the CALD program decodes the interrupt message which it

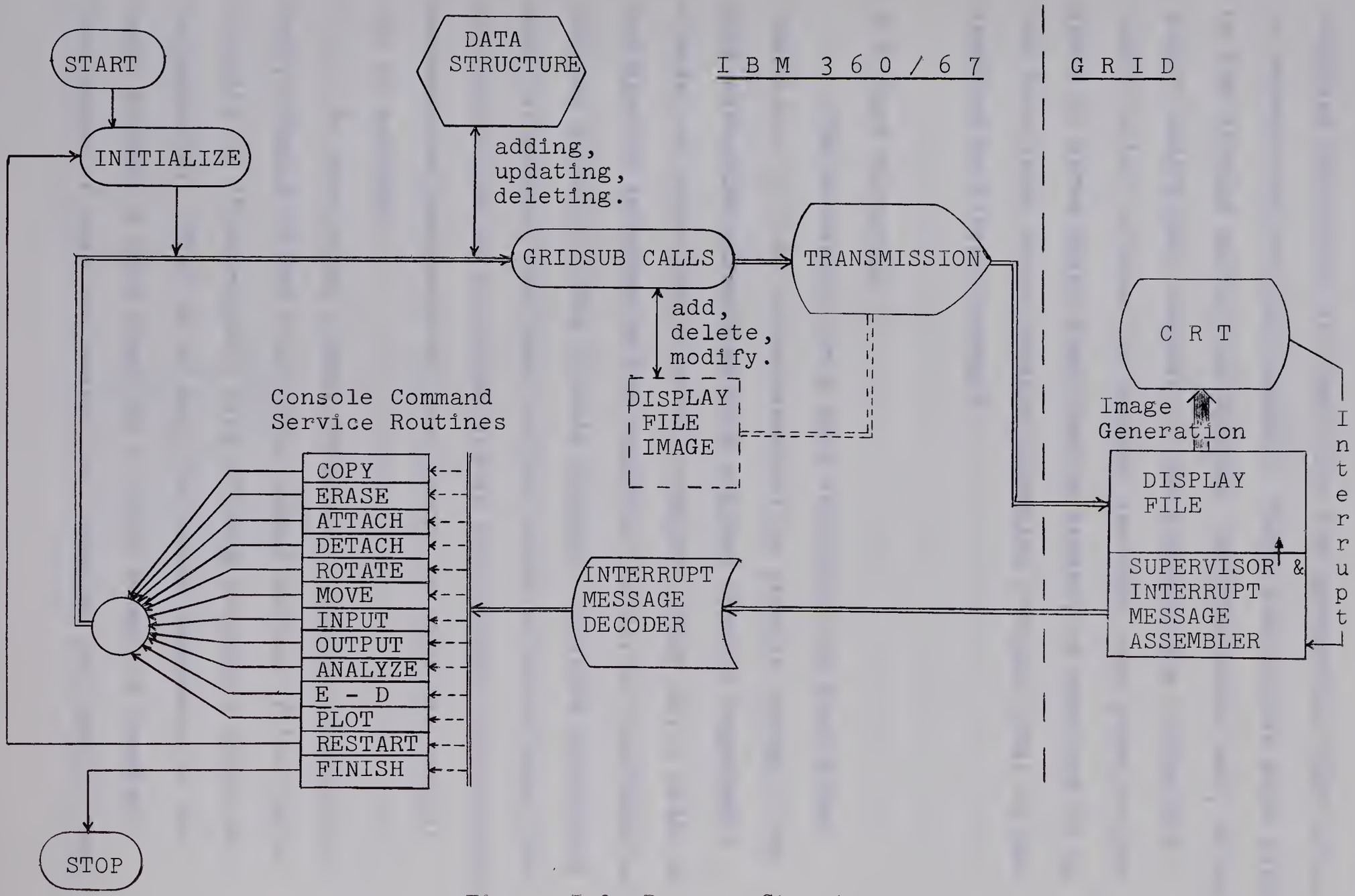


Figure 5.9 Program Structure

receives from GRID; it then calls the appropriate subroutine to service the console command. These subroutines make calls to the display subroutine package, GRIDSUB, which add, delete and/or modify the contents of the display file. From the user's point of view the changes can take such forms as new gates or wires being displayed or erased, or they may be in the form of an error message informing the user that he has executed an illegal command.

5.4 Data Structure

The necessity for a data structure has been shown (Section 2.5, 3.4) to be essential in circuit design. The data structure associated with a given circuit represents a model of that circuit in the computer; that is, a model of the circuit in terms of the function the circuit performs as well as a model of the circuit diagram. Besides containing some information pertinent to the coding of the circuit, the data structure must include all of the relevant associations between the components of the circuit so that the circuit can be analyzed.

In developing a data structure for CALD it was felt that, because of the relatively simple nature of the logic circuits, a "fixed-block" type of data structure could be incorporated. That is to say, for every component of the logic circuit a data block of a fixed length is created. The length of each and every block remains the same for the

duration of the block. This type of structure differs from a "variable-block" type in which the dimensions of the blocks can vary dynamically as different demands are placed upon it.

To make the data structure powerful, an essential set of pointers form the nucleus of the model. These pointers indicate the associations between the circuit components and also facilitate rapid referencing to the circuit components by avoiding numerous time-consuming sequential searches.

Essentially there are two types of blocks in the data structure: gate blocks and wire blocks. The first type are called LOGBLOC (logical gate blocks) and the second type are called CONBLOC (connector blocks; wires). The blocks are essentially "n-component elements" (Ross et al. 1963) where an n-component element is defined as a single unit of information about a problem, which specifies in each of its components one attribute or property of the element. The blocks together with their pointers form a structure called a "plex" (Ross et al. 1963), where a plex is an interconnected set of n-component elements.

A LOGBLOC is a 10-component element as illustrated in Figure 5.10. For each gate and node created there is an equivalent LOGBLOC.

TYPE	O.F.
X	Y
R	OUT
PT1	PT2
IN1	IN2

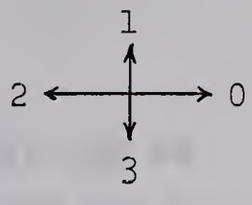
Figure 5.10 LOGBLOC.

TYPE - is a unique number equal to BN*100+ID, where BN (<50) designates the type of gate (AND/OR/NOT), and ID is an identification number between 0 and 63.

O.F. - is an output flag; designates if the output of the gate is attached (wired) to another gate, and, if so, it indicates the number of inputs to which it is attached.

X,Y - indicates the coordinate position of the gate on the screen. The units are in 100ths of inches.

R - contains the orientation of the gate



OUT - contains the output logical value of the gate when the circuit has been analyzed.

PT1,PT2- input pointers; indicate if the inputs are attached and if they are, then PT1, and PT2 will point to the respective LOGBLOC representing the gate from which the current gate derives its input.

IN1,IN2- contain the logical value of the inputs.

A CONBLOC is again a 10-component element as shown in Figure 5.11. For each wire there is an equivalent CONBLOC. Associated with the whole set of CONBLOCs are two lists: CHAIN and WIRE. CHAIN is a string of pointers which interconnect all of the wires (CONBLOCs). The CHAIN is useful when CALD requires to perform an operation on all of the wires. For example in the E-D command, the CHAIN facilitates

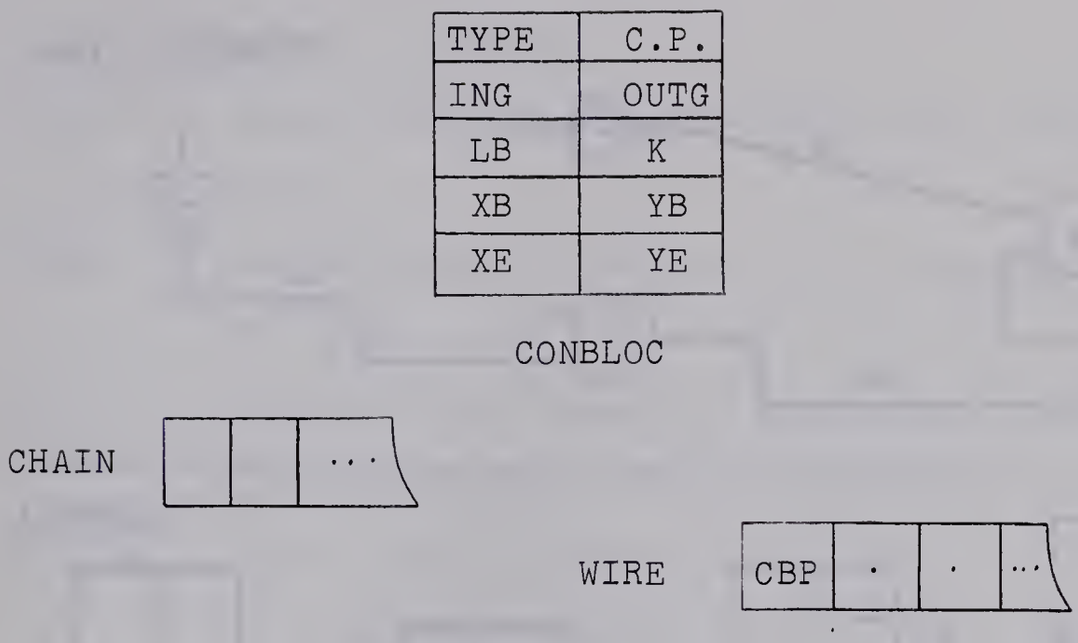


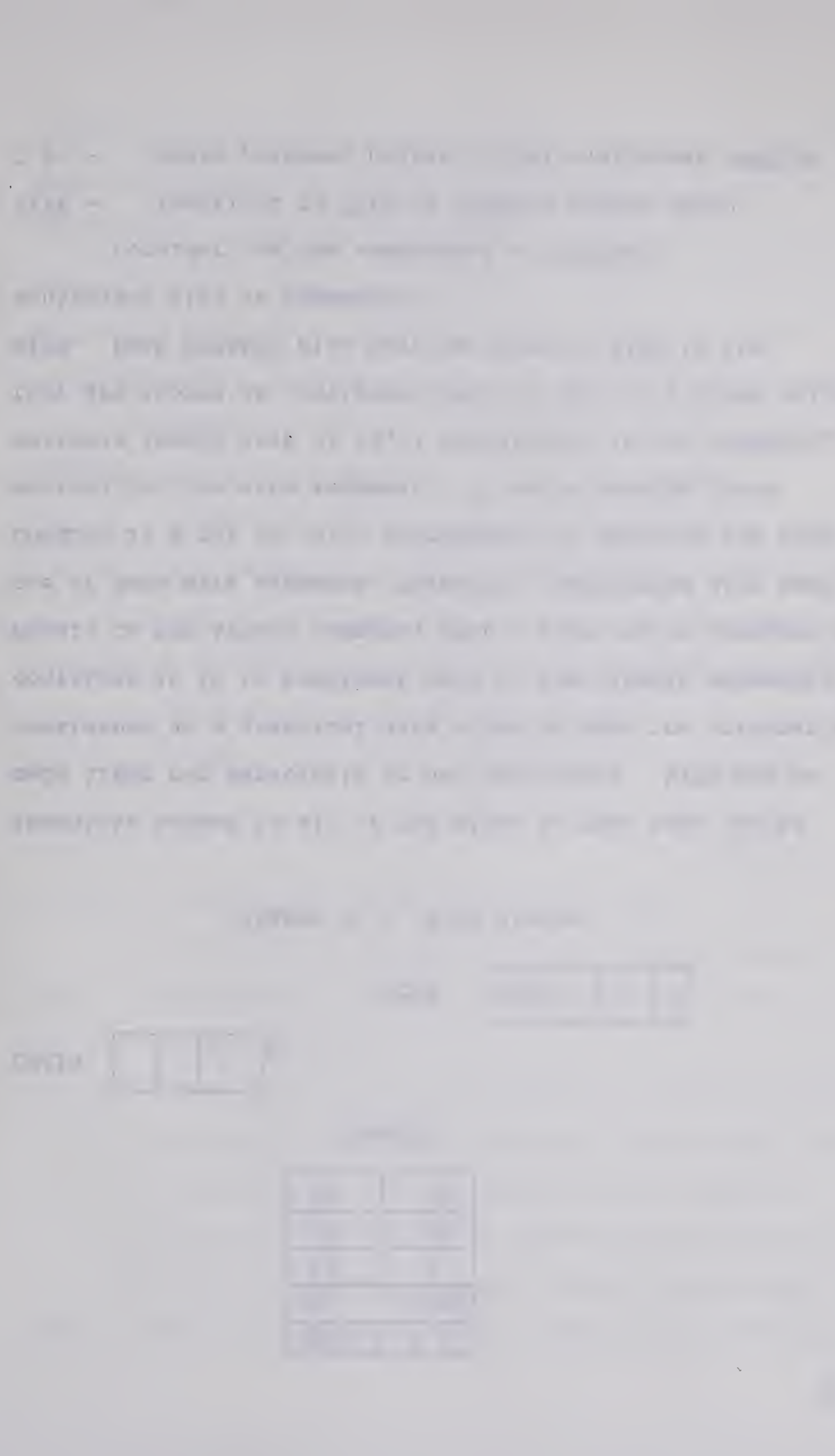
Figure 5.11 Wire Blocks

immediate access to all of the wires so that they can be made light pen detectable or not detectable. WIRE can be considered as a graphical data block in that the information contained in it is pertinent only to the circuit schematic. Recall in the ATTACH command, that a wire can be composed of one or more wire segments; therefore, associated with each CONBLOC is a set of (x,y) coordinates to describe the line vectors for the wire segments. To avoid storing these variable length sets of (x,y) coordinates in the CONBLOCs, they are stored as contiguous sets of data in a block called WIRE. Each CONBLOC will have an entry in WIRE if its equivalent wire is segmented.

Consider now the components of CONBLOC.

TYPE - identical to TYPE in LOGBLOC except BN>50.

C.P. - chain pointer; points to the equivalent CONBLOC



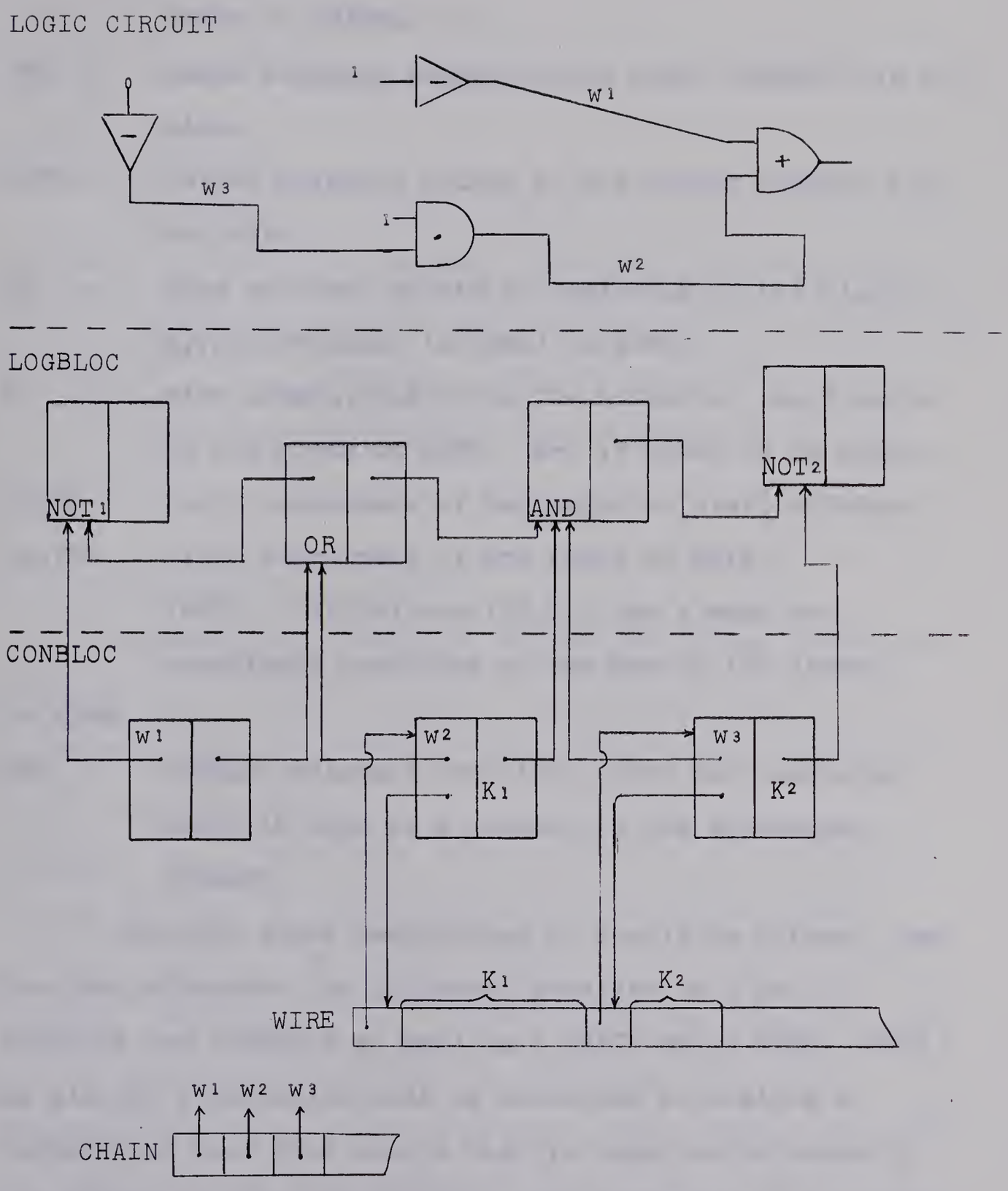


Figure 5.12 A logic circuit with its data structure

entry in CHAIN.

ING - input pointer; points to the input LOGBLOC for the wire.

OUTG - output pointer; points to the output LOGBLOC for the wire.

LB - wire pointer; points to beginning of the block of (x,y) coordinates (if any) in WIRE.

K - wire length; specifies the number of (x,y) pairs in the block in WIRE. K=0 if there is no entry.

XB,YB- (x,y) coordinate of beginning or start of wire.

XE,YE- (x,y) coordinate of end point of wire.

(NOTE: (XB,YB) and (XE,YE) are always the coordinate positions of the end of I/O leads).

In WIRE:

CBP - CONBLOC pointer; the first entry for each wire entry in WIRE is a pointer to the equivalent CONBLOC.

From the above description it should be evident that the data structure for a circuit consists of a set of LOGBLOCs and CONBLOCs as well as a CHAIN and a WIRE. This is all the information that is necessary to analyze a circuit and also this is all that is required to store if one wishes to file the circuit. The schematic of a circuit can easily be reconstructed from this data structure.

An example of a small logic circuit plus its equivalent data structure is shown in Figure 5.12.

CHAPTER VI

CALD EVALUATION

6.1 Introduction

The intention in this chapter is to appraise the CALD system as it has been realized. The fulfillment by CALD of the requirements of an on-line design system, as presented in Chapter II, is discussed. The data structure in CALD is also assessed and some suggestions for improvements are presented.

6.2 Interaction Requirements

Certain requirements of an on-line design system have been set out in Section 2.3.3. A reasonable approach to evaluating CALD is to discuss the system in the context of these requirements.

Editing: One of the basic features built into the CALD system provides for modifying circuit components which already appear on the screen. Gates can be copied, moved, rotated, and erased; they can be attached or detached. All of the actions mentioned can be considered part of an editing function. In the context of logic design, the scope of these actions is adequate and thus it is reasonable to say that the editing requirement is satisfied.

Output: In that CALD uses a graphical display for output, this requirement is easily met. All circuit schematics are

displayed and all inputs and outputs are displayed at their relevant locations. Since the output from any circuit analysis is very simple (1 or 0) the displaying of results presents no problems. Hard copies of the output are available through use of a digital plotter.

Definitional: In systems similar to CALD, a definitional facility could typically provide for creating new design components or for creating new edit commands. At present CALD does not meet this requirement. To provide a definitional facility is far from being a trivial task, and so it was not provided at this time. However, in a later version of CALD which incorporated the "black box" option, the ability to create new design components could be realized. For example, suppose a designer wishes to build a circuit out of NOR and NAND logic components. He could build each of these components using the AND, OR, and NOT gates, "black box" them to be in the form of single pictorial components, and then use them as basic components for his design.

Informational: The informational service provided by CALD is almost non-existent. Because of the relative simplicity of the design technique, it was felt that the informational requirement was not essential. Effectively, CALD operates in only one state; this is different from some other systems such as LDB which is operational in one of several modes. As a result, an information service which can indicate the state or mode the system is in, is not necessary. Also

because of the nature of the design actions, that is, each design action is equivalent to one console command message, CALD messages indicating "what to do next" would not be meaningful.

Diagnostic: All console command messages which are sent to CALD for interpretation are checked for their validity. Design actions can be checked for their validity, but console command actions cannot be checked. If the user performs some invalid action then an appropriate error message is set blinking on the screen. The diagnostic requirement is thus provided satisfactorily in CALD.

6.3 Data Structure

The purpose of any logic circuit data structure is to model the logic circuit which has been created on the CRT. A good model is one which can be conveniently accessed and easily manipulated. The data structure in CALD is quite straight forward and yet it accommodates all of the demands made upon it by circuit design manipulations. With the sets of pointers inherent in the data structure, access to associated components of the structure is easily facilitated. The fact that the circuit analysis program is a relatively short and uncomplicated algorithm exemplifies the appropriateness of the chosen structure. Also the completeness of the structure is illustrated by the fact that a filed circuit diagram can be entirely reconstructed from the data structure

such that further circuit design, analysis, and testing can be carried out.

6.4 Extensions and Suggestions

6.4.1 Console Command Language

The versatility of the console command language could be enhanced by permitting multiple "COPY's" and "ATTACH's". This is somewhat difficult to implement because both of these design actions can be constructed of variable length console command messages. One approach would be to consider selections of "COPY" and "ATTACH" as entering into "copy mode" and "attach mode" respectively. A mode would be initialized by selecting the appropriate "mode light button". The system would then continue in this mode until another mode button was selected.

For example: Consider the two following sequences of design actions. The first sequence exemplifies the system as it now stands; the second sequence is the new proposal.

Present:

```
COPY $ AND $ XY
COPY $ OR $ XY
COPY $ AND $ XY $ R2
COPY $ NOT $ XY $ R3
ATTACH $ I/O lead $ I/O lead
ATTACH $ I/O lead $ XY $ I/O lead
ATTACH $ I/O lead $ XY $ XY $ XY $ I/O lead
```

```
        ATTACH $ I/O lead $ I/O lead
        ERASE $ OR $ XY
```

Proposed:

```
        COPY $ AND $ XY
               OR $ XY
               AND $ XY $ R2
               NOT $ XY $ R3
        ATTACH $ I/O lead $ I/O lead
               I/O lead $ XY $ I/O lead
               I/O lead $ XY $ XY $ XY $ I/O lead
               I/O lead $ I/O lead
        ERASE $ OR $ XY
```

At present the MOVE command permits unattached gates to be "moved" to any position on the screen. It would be very useful to allow complete or partially complete circuits to be "moved" to new positions on the screen. Thus by issuing a MOVE with one gate as an operand, all attached components would also move with their relative positions unchanged. At present, if a circuit is being constructed and if some work area becomes crowded the designer has no recourse but to rebuild his circuit if he wishes to remedy the situation.

6.4.2 Filing and Labelling

The proposal for a CALD system called for the "filing"

or "storing" of circuits and also the subsequent retrieval of these circuits. The data structure as it now stands contains all of the information necessary for this type of action. The only feature not existing is a method of referencing a circuit once it has been created. This could be resolved by labelling all circuits before they are filed. GRID has an alphanumeric console so that, at appropriate times, circuits could be labelled by typing in characters from the keyboard. All circuits could then be stored and retrieved by referencing the circuit labels. Extending this further, then, it would be useful to have a circuit library to which the designer could add, and from which he could retrieve circuits. At any point then the user should be able to ask for a catalogued list of all of the circuits in the library. Extending this even further, one can envision an environment with many CALD users with say each user having his own library and perhaps there could be a shared library in which users could retrieve circuits but not file circuits.

6.4.3 Black Box Circuits

As has been proposed previously, the ability to reduce schematically logic circuit diagrams to simple components, black boxes, is essential if circuits are going to be designed with any degree of sophistication. Using the light pen, the black box can be constructed and then after all of

the inputs and outputs have been designated, the black box could be called to replace the schematic diagram of the logic circuit. The data structure would remain unchanged except a graphic data block would have to be added which would describe pictorially the black box. If at any time the original circuit schematic was requested then it could easily be retrieved from the data structure.

In developing the black box discussion further, consider the black box feature representing part of a concept of taking a complex graphic entity (which may represent a simple or complex function) and reducing it to a much simpler graphic entity which still represents the same function. This concept could be classified as "graphic reduction". Graphic reduction offers a number of advantages and perhaps is a partial answer to one of the problems of computer graphics: data management. The obvious advantage of graphic reduction is that by reducing the diagram of a design object the design area (screen) can become less cluttered and more objects can occupy the screen at the same time. With regard to data management, commonly in the design of some required object, tremendous graphic files and data structures are created which represent the element in two ways: schematically and functionally. The complexities of manipulating these large files are non-trivial and so a reduction of the files would certainly be beneficial. After the files have been created, it is quite possible that the design object could

be represented schematically by a much simpler entity. (Recall for any given design object that there is usually only one functional representation but an infinite number of pictorial representations). Originally a complex diagram was probably necessary to facilitate the building of an exact model (functional representation) for the object, but after this has been created then the original schematic can be discarded for a more simple diagram.

Furthermore, a simplification of the functional representation of a design object can possibly be accomplished. In a given design object the equivalent data structure does not usually represent directly the function of the object; the functional representation can generally be considered to be latent in the structure. However, after the object has been created, it is perhaps possible to explicitly represent the function of the object with a small number (one or two) of data structure blocks rather than representing the object implicity with a large number of interrelated blocks. This type of reduction could again facilitate considerable savings in the storage requirements for the data structure and also aid in minimizing the problems in data structure manipulation.

The effort in performing data structure reduction could be considerable, however, it should certainly be worthwhile if significant reductions can be made in the files and structures which have to be manipulated.

6.4.4 Windowing

Windowing is a feature which could permit the design of a circuit on an area larger than the screen area. The CRT then acts like a window looking down onto selected parts of the drawing area or board. This feature is usually essential if a designer wishes to design any big circuits, for it is unreasonable to assume that all of his circuits can be fitted on a 12 x 12" drawing area. CALD does not incorporate this feature and it is not proposed for the near future for it is felt that if the user had the "black box" facility, then windowing would not be so essential. If the designer were building a large sophisticated circuit he could build it in parts, and then black box these parts which would then effectively compress his circuit schematics. If at any later time he wishes to specifically interrogate these circuit parts then the original circuit schematics could be easily retrieved.

6.5 Conclusion

Within the objectives of the initial system, CALD has been a success. CALD does provide a system which enables a user to design simple logic circuits and test them. More important than this is that CALD does provide a promising basis for a more sophisticated system. All of the suggestions previously mentioned can definitely be incorporated into the present system. The next major step in building a

complete CALD system is to provide for building of sequential circuits; this then will allow for useful design of computer logic components.

CHAPTER VII

IMPLICATIONS FOR A SYNTAX DIRECTED SYSTEM

7.1 Introduction

The approach taken in developing the CALD system, and systems of similar nature has been generally of a heuristic nature. Systems of this type can be clumsy in terms of checking for validity of inputs, analyzing inputs, and analyzing accumulated results of the system.

Also, previously and currently developed systems offer little as far as generalized systems go. The design of any given system has tended to have no applicability from one project to the next. As a result of this, in constructing CALD, thought has been given to the possibility of building a system in a more or less formal manner. This has led to some deliberation on the feasibility of a syntax-directed design system. Whether such a system would be of direct, practical use is undetermined; however, its specification and possible implementation should give considerable insight into what is precisely involved in computer-aided design.

Consider the whole design system to be composed of three parts: communication, circuit modeling, and circuit model analysis. The communication system permits the designer to communicate with the modeling system; that is, his design actions, transmitted via the console command language, are translated into meaningful expressions for

the modeling system. The logic circuit modeling system builds and manipulates a data structure to form a model of the logic circuit as described by the designer. The analysis system then applies an algorithm to the data structure to produce a result.

In that all of these three systems must interact with each other, the possibility exists that in formalizing each system or parts of each, a more expedient type of communication between the systems could be facilitated.

Some of these thoughts have been suggested by the work of Shalla (Shalla 1966) where he developed a program for automatic circuit analysis which made use of a formalization of an electronic circuit. He defined all logic circuits as belonging to a set of circuits defined by three formal rules. Also with regard to natural language and pictures, Kirsch (Kirsch 1964) has argued that

> "from the point of view of computer information processing, the important fact about natural language text and pictures is that both have a syntactic structure which is capable of being used for purposes of interpreting the information within a data processing system."

It is the purpose of this chapter to entertain the possibility of a syntax-directed design system without being overly concerned with the practicability of such a project.

7.2 Console Command Language Specification

Consider the main object of the design system, the

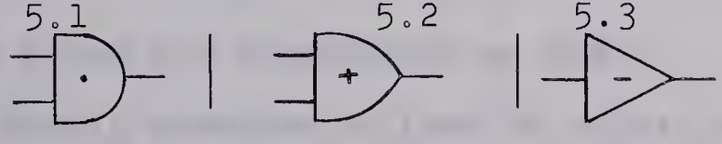
logic circuit. There is certainly evidence of some type of structuring in a logic circuit, for there are allowed as well as disallowed connections between the logic components. Moreover, it follows that the actions of the designer must concur with the specifications of a logic circuit. It is possible to develop a command language specification which would define all console commands which can be issued by the designer. Consider the following attempt using only a subset of the allowable commands. (ϕ will be used here and throughout this chapter to specify a null set).

1.1 1.2
<console command> → <copy command>|<attach command>

2.1 2.2
<copy command> → COPY<component>XY|COPY XY <component>

3.1
<attach command> → ATTACH <xy seg>

4.1 4.2
<xy seg> → ϕ |XY <xy seg>

5.1 5.2 5.3
<component> → 

In the above command language specification, the gates, inputs, and outputs are considered as basic symbols. It is clear that an extension to a BNF-like specification is needed to define properly a language of this type. The element XY is itself not really a basic symbol but refers to

any position on the screen. The pictorial entities, which are basic symbols are displayed on the screen. The "input" of a basic symbol is effected by the action of pointing to that symbol. The specification shows the allowable choices, with the juxtaposition of the categories and basic symbols showing the order of selecting various command components. The command language is clearly a string language, and its definition by a generative grammar is obviously reasonable.

Parallel with the definition of the command language must be the definition of pictorial primitives. These could be defined formally using techniques similar to those which Ledley (Ledley-2 1965) used for describing chromosomes or those which Narasimhan (Narasimhan 1966) used for describing bubble chamber pictures.

7.3 Logic Circuit Specification

Specifying a "syntax" of logic circuits presents special problems in that the relationships between components of logic circuits are more complex than in conventional string languages. If the gates are considered as basic symbols, then the syntax should show the allowable relationships between the gates. Ledley (Ledley-1 1962), in giving his syntax specification for a house, made use of "syntactic relationships" between the basic symbols. This type of device could be useful at this point. Consider the following syntactic relationship definition:

→| - 'is input to'

Further, it is necessary to introduce the notion of "attribute" as defined by Narasimhan (Narasimhan 1966), that is, an attribute is a function defined over an object. Denote an attribute by γ( ). Thus <a>γ(Attrib) denotes that the category <a> has the attribute Attrib.

Also before giving a syntax specification for logic circuits the defining of a meta-linguistic device would be useful:

n↑<a> - 'denotes an integer n number of categories <a>'

The syntax specification is then:

1.1
<logic circuit> → <logic gate>γ(OUTPUT)

2.1 2.2 2.3
<logic gate> → <and gate> | <or gate> | <not gate>

3.1
<and gate> → 2↑<input> →| AND

4.1
<or gate> → 2↑<input> →| OR

5.1
<not gate> → <input> →| NOT

6.1 6.2
<input> → <logic circuit> <wire> | <logic variable>

7.1 7.2
<wire> → ϕ | WIRE

8.1
<logic variable> → V

This specification for a logic circuit can be considered analogous to a generative grammar in that by using this circuit "grammar" a description of all well-formed logic circuits can be generated. Considering the specification as a set of productions then perhaps they can be used to generate data structures for logic circuits. By assigning "meanings", in terms of data structure development, the above specification can be used to generate any or all data structure models of logic circuits. The following is a set of semantics for generating the components for the data structure.

Rule No.	Semantic Interpretation
1.1	Create LOGBLOC, set TYPE, inputs fields
2.1	TYPE←AND + ID
2.2	TYPE←OR + ID
2.3	TYPE←NOT + ID
3.1	Create PT1, PT2, fields
4.1	Create PT1, PT2, fields
5.1	Create PT1 fields
6.1	Create CONBLOC, set LB and K, LOGBLOC pointers, CONBLOC pointers
6.2	Set logical value flag
7.1	K ← 0, LB←LB
7.2	Set XY entry in WIRE, LB←LB+2 K←K+1
8.1	Set logical value

7.4 Command Language/Productions Mapping

Consider what has been presented at this point in this discussion. In the communication system there is a console command language specification, and in the modeling system there is a logic circuit specification with a set of semantics for generating the data structure. To put these new developments to use to aid the effective interaction between the two, a mapping of the console command language onto the productions is necessary. Then, when a designer executes a console command, the system can analyze the command message to generate the rule numbers for the command. These rule numbers can then be mapped onto the data structure productions to produce the proper components for the circuit model.

Consider the following mapping:

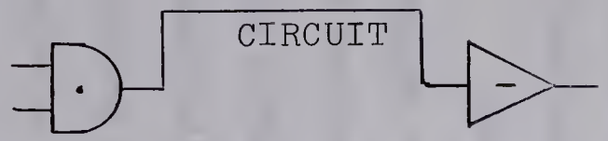
Command Language Rule No.	Production Rule No(s).
1.1	ϕ
1.2	ϕ
2.1	1.1
2.2	1.1
3.1	6.1
4.1	7.1
4.2	7.2
5.1	2.1, 3.1
5.2	2.2, 4.1
5.3	2.3, 5.1

To see how this system might operate, consider the tabulated procedure for building a small circuit which is illustrated in Example 1. The data structure development is "executed" in an order corresponding to a bottom-up labelling of the tree resulting from the parse of the input commands. Although the example lacks completeness it does show that the mapping from a command language syntax to rules for generating a data structure is at least feasible.

7.5 Implications

The implications of a syntax-directed system similar to the type suggested is appealing for a number of reasons. Consider the console command language. If a formal specification for the command language were available, then all the command messages could be checked for validity by parsing the input strings of commands to obtain the tree for each command. This parsing could be accomplished by a program similar to the ANALYZER in Cheatham and Sattley's syntax-directed compiler. (Cheatham et al. 1964). If the parse were successful, then the output from the analyzer would be a string of rule numbers (corresponding to tree nodes) which would be passed to the next stage of the system for generating the data structures. In a design configuration similar to CALD it would be advantageous if the "analyzer" could reside in the small graphical subsystem. Then all command messages could be checked immediately without going

EXAMPLE 1.



STEP	DESIGNER ACTION	COMMAND LANGUAGE RULE NO.	CIRCUIT GRAMMAR RULE NO.	DATA STRUCTURE DEVELOPMENT
1	With light pen, pick: COPY $ ⊐ $ XY $	5.1 2.1 1.1	2.1 3.1 1.1 ϕ	TYPE←AND+ID PT1, PT2 LOGBLOC: AND / PT1 PT2
2	With light pen, pick: COPY $ XY $ ▷ $	5.3 2.1 1.1	2.3 5.1 1.1 ϕ	TYPE←NOT+ID PT1 LOGBLOC: NOT / PT1 LOGBLOC: AND / PT1 PT2
3	With light pen, pick: ATTACH $ – $ XY $ XY $ –	4.2 4.2 3.1 1.2	7.2 7.2 6.1 ϕ	K←1 LB←3 K←2 LB←5 LB´ ← LB-2*K CONBLOC: LB / K LOGBLOC: NOT LOGBLOC: AND / PT1 PT2 WIRE: X Y X Y

to the S/360. Moreover, it would only be necessary to transmit the string of rule numbers to the S/360 when a valid command message was formed. This could facilitate a faster response time by transmitting less information and also a more economic use of S/360, by having it only contend with valid data. Another advantage to a system of this nature is the ease with which the command language could be changed and equivalent alterations made to the data structure productions. All that would be necessary is to change some or all of the entries in the syntax tables and the semantics tables. No adjustments would be necessary in the programming of the "analyzers", etc. Also, since the circuit "grammar" defines all well formed circuits, and since the data structure for a particular circuit is generated by this "grammar", then checking for completeness and correctness is facilitated.

In conclusion, the type of syntax-directed system which has been idealized here shows analogies with a table-driven compiler of Feldman (Feldman 1966), see Figure 7.1.

The language L corresponds to the console command language with the source code being console command messages; and machine code generation paralleling the generation of data structure components. This analogy suggests the possibility that the ultimate goal that should be sought in building a design project is to produce a system which can handle a variety of applications. That is, if for a given application the system was provided with a command language

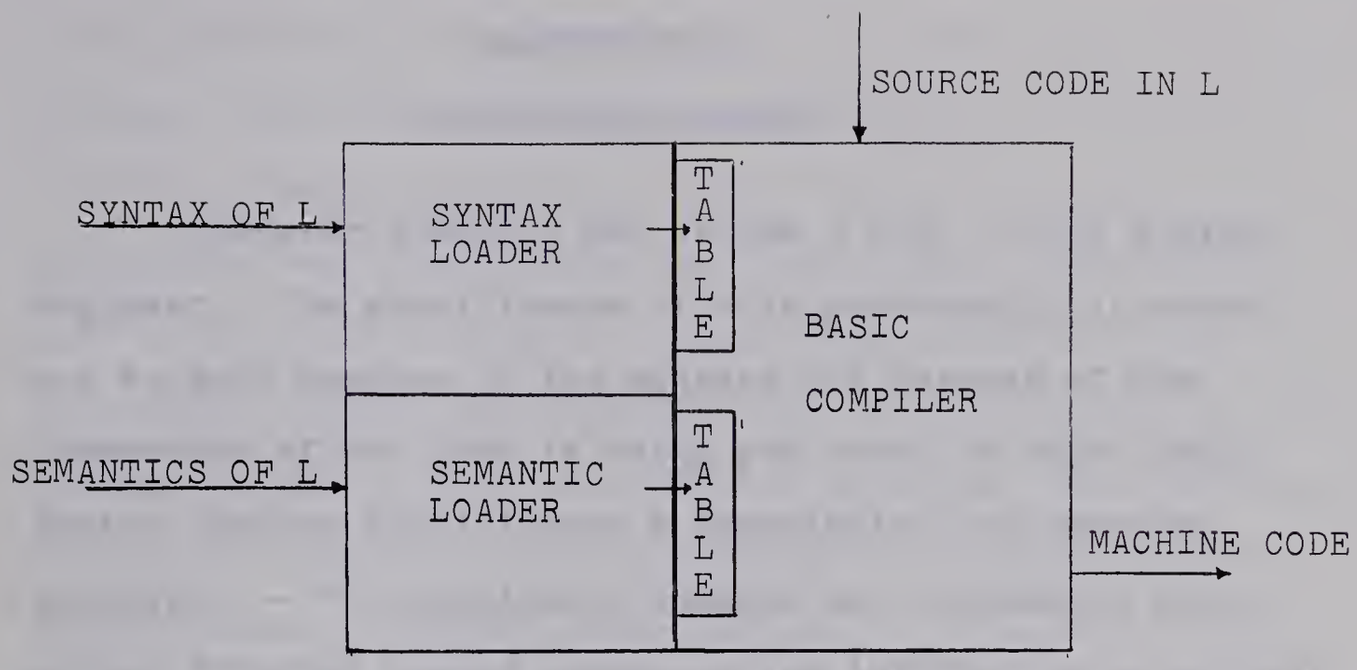


Figure 7.1 A Compiler-compiler (Feldman 1966)

specification for the problem as well as a table of semantics to generate data structures, the system could perform the required function for the application. A system of this nature could be labelled as a General Purpose Interactive Graphics Processor.

CHAPTER VIII

CONCLUDING REMARKS

Computer graphics has become a tool of the design engineer. The significance of this partnership is noted not so much because of its success but because of the tremendous effort that is being put forth in this field. Fetter (Fetter 1967) states a description for computer graphics: -- "A consciously managed and documented technology directed toward communicating information accurately and descriptively". At this time the author believes that Fetter's remarks tend to be more of a goal for computer graphics, than a definition.

Computer-aided design, particuarly design of electronic circuits has been reviewed and investigated in this thesis. As indicated in the introduction, the success of much of the previous work has been less than overwhelming. It is perhaps unfair to be too critical of the development of interactive graphics, for it is being compared with the unparalleled success of digital computer development in the last fifteen years.

However, the results have not been totally discouraging. The achievements of such systems as OLCA and LDB do generate some optimism. These systems were developed as experimental projects to determine if on-line design systems using graphical displays were feasible. They were successful in this objective; also they revealed some of the

difficulties which were inherent in building truly effective systems. The need for further investigations in areas of console command languages, program organization, and data structures were demonstrated by their work. Although past work in interactive graphic design systems has not solved the problem areas, they have at least been clarified to some degree. The lack of preciseness in defining the requirements of on-line design systems has assisted in compounding the difficulties of building such systems. The contributions made in previous work towards setting these requirements have been valuable.

With the problems of interactive design systems at hand, the CALD system was proposed and built to further investigations in this field. The CALD system was designed to be simple so that due emphasis could be placed on the problems of interactive graphics rather than the computing complexities involved in circuit analysis.

Out of the CALD project came the realization that many of the previous achievements had been accomplished in an ad hoc manner. The work of a given group of people was forming no basis for, or portion of, the next group's effort in computer-aided design. Some suggestions were thus presented on the possibility of building formalized systems which could be general enough to handle a variety of applications. An investigation into the worth of these suggestions would surely be of some benefit.

The computer world is exerting a lot of effort in the realm of interactive graphical design systems. If success is slow in coming, then why does the exertion continue to be so intense? It must be because the applications community is striving to get away from paper tapes, punched cards, and typewriter terminals. The users want to speak their own language which is most often schematic and so the apparent solution is computer graphics. Therefore the work in this field must go on so that man can master the machine, for as long as man speaks the machine's language and not his own, then the machine is master.

BIBLIOGRAPHY

Bobrow, D.G. and B. Raphael, 1964. "A Comparison of List-Processing Computer Languages", C.A.C.M., Vol. 7, No. 4, pp. 231-240.

Cancro, R., and D.L. Slotnick, 1968. "Some Thoughts on Displays", Emerging Concepts in Computer Graphics, D. Secrest and J. Nievergelt (editors), W.A. Benjamin Inc. New York. pp. 85-99.

Chasen, S.H., 1968. "Experiences in Application of Interactive Graphics". Emerging Concepts in Computer Graphics, D. Secrest and J. Nievergelt (editors), W.A. Benjamin, Inc. New York. pp. 255-309.

Cheatham, Jr. T.E., and K. Sattley, 1964. "Syntax-Directed Compiling", Proc. Eastern Joint Computer Conference, AFIPS, Vol. 25, pp. 31-57.

Christiansen, D., 1966. "Computer-aided design: part 1 - The man-machine merger", Electronics, Vol. 39 pt. 3, September 19, pp. 110-123.

CDC, 1968. GRID Engineering Specification, Data Display Division, CDC.

Coons, S.T., 1963. "An Outline of the Requirements for a Computer-Aided Design System", Proc. AFIPS, Vol. 23, pp. 299-304.

Cross, P., 1967. "A logic drawing board for the PDP/340", The Computer Bulletin, Vol. 11, No. 3., pp. 237-245.

Dawson, D.F., F.F. Kuo, and W.G. Magnuson, 1967. "Computer-Aided Design of Electronic Circuits, A User's Viewpoint", Proc. IEEE, Vol. 55 No. 11, pp. 1946-1953.

Dertouzos, M.L., 1967. "CIRCAL: On-Line Circuit Design", Proc. IEEE, Vol. 55, No. 5, pp. 637-654.

Dertouzos, M.L., 1967. "An Introduction to On-Line Circuit Design", Proc. IEEE, Vol. 55, No. 11, pp. 1961-1970.

Dertouzos, M.L., 1968. "Man-Machine Interaction in Circuit Design", Proc. Hawaii International Conference on System Sciences, B.K. Kinariwala and F.F. Kuo, (ed.), pp. 100-102.

Evans, D.S. and J. Katzenelson, 1967. "Data Structure and Man-Machine Communication for Network Problems", Proc. IEEE, Vol. 55, No. 7, pp. 1135-1144.

Feldman, J.A. 1966. "A Formal Semantics for Computer-Languages and its Application in a Compiler-Compiler", C.A.C.M., Vol. 9, No. 1, pp. 3-9.

Fetter, W.A. 1968. "Computer Graphics", Emerging Concepts in Computer Graphics, D. Secrest and J. Nievergelt (editors), W.A. Benjamin, Inc. New York. pp. 397-418.

Fisk, C.J., D.L. Caskey, and L.E. West, 1967. "ACCEL: Automated Circuit Card Etching Layout", Proc. IEEE, Vol. 55 No. 11, pp. 1971-1982.

Gray, J.C., 1967. "Compound data structure for computer-aided design; a survey", Proc. ACM Nat. Conf. 22nd., pp. 355-365.

Hellerman, H., 1967. Digital Computer System Principles, McGraw-Hill Book Co. New York, New York.

Huen, W.H., 1969. A Graphical Display Subroutine Package, (M.Sc. Thesis), Department of Computing Science, University of Alberta, Edmonton.

Iverson, K.E., 1967. A Programming Language, John Wiley, New York.

Johnson, T.E., 1963. "Sketchpad III, A Computer Program for Drawing in Three Dimensions", Proc. AFIPS, Vol. 23, pp. 347-353.

Kirsch, R.A., 1964. "Computer Interpretation of English Text and Picture Patterns", Trans. IEEE EC 13, pp. 363-376.

Kuo, F.F., 1966. "Network Analysis by Digital Computer", Proc. IEEE, Vol. 54, No. 6, pp. 820-829.

Kuo, F.F., W.G. Magnuson, and W.J. Walsh, 1969. "Computer Graphics in Electronic Design", Datamation, Vol. 15 No. 3, pp. 71-79.

Ledley, R.S., 1962. Programming and Utilizing Digital Computers, McGraw-Hill, New York 1962.

Ledley, R.S., 1965. Use of Computers in Biology and Medicine, McGraw-Hill Book Co. New York, New York.

Lee, H.B., and R.D. Thorton, 1967. "An Experiment in Computer-Aided Education", Proc. IEEE, Vol. 55, No. 11, pp. 2001-2005.

Lewin, M.H., 1967. "An Introduction to Computer Graphic Terminals", Proc. IEEE, Vol. 55, No. 9, pp. 1544-1552.

Logan, J.R., 1968. "Computer Graphics in Electronic Circuit Design", Computers and Automation, Vol. 17, No. 11, pp. 28-30.

Mittleman, J., 1969. "Computer Aided Design, act two: Admission price exceeds forecasts", Electronics, Vol. 42, No. 12, pp. 90-98.

Murray-Lasso, M.A., 1968. "On-line Circuit Design on Commercial Time Shared Systems", Chapter 8, Computer-Aided Integrated Circuit Design, G.J. Herskowitz, (editor), McGraw-Hill Book Co., New York, New York, pp. 280-322.

Narasimhan R., 1966. "Syntax-Directed Interpretation of Classes of Pictures", C.A.C.M., Vol. 9, No. 3, pp. 166-173.

Newman, W.M., 1966. "An Experimental Program for Architectural Design", Computer Journal, Vol. 9, No. 1, pp. 21-26.

Ninke, W.H., 1965. "Graphic 1-A Remote Graphical Display Console System", Proc. FJCC, Vol. 27, Part 1, pp. 839-892.

Ninke, W.H., 1966. "Man-Computer Graphical Communication", System Analysis by Digital Computer, F.F. Kuo and J.F. Kaiser, (eds.), John Wiley and Sons, Inc., New York, pp. 395-432.

Parker, D.B., 1967. "Solving Design Problems in Graphics Dialogue", On-Line Computing, W.J. Karplus, Ed., New York: McGraw-Hill, pp. 1-12.

Phister, M., 1960. Logical Design of Digital Computers, John Wiley and Sons, Inc., April 1960.

Prince, M.D., 1966. "Man-Computer Graphics for Computer-Aided Design", Proc. IEEE, Vol. 54, No. 12, pp. 1698-1708.

Roberts, L.G., 1964. Graphical Communications and Control Languages, M.I.T. Lincoln Lab., Reprints MS1173.

Ross, D.T., and J.E. Rodriguez, 1963. "Theoretical Foundations for a Computer-Aided Design System", Proc. AFIPS, Vol. 23, pp. 305-322.

Sedore, S.R., 1966. "Computers Aid Circuit Design", Electronic Design, Vol. 14 No. 1, pp. 96-99.

Shalla, L., 1966. "Automatic Analysis of Electronic Digital Circuits Using List Processing", C.A.C.M., Vol. 9, No. 5, pp. 372-380.

So, H.C., 1966. "Analysis and Iterative Design of Networks Using On-Line Simulation", System Analysis by Digital Computer, F.F. Kuo and J.F. Daiser (eds), John Wiley and Sons, Inc., New York, pp. 34-58.

So, H.C. 1967. "OLCA: On-Line Circuit Analysis System", Proc. IEEE, Vol. 55 No. 11, pp. 1954-1961.

Spitalny, A. and M.J. Goldberg, 1967. "On-Line Graphics Applied to Layout Design of Integrated Circuits", Proc. IEEE, Vol. 55, No. 11, pp. 1982-1988.

Stotz, R., 1963. "Man-Machine Console Facilities for Computer-Aided Design", Proc. AFIPS, Vol. 23, pp. 323-328.

Sutherland, I.E., 1963. Sketchpad: A Man-Machine Graphical Communication System, Tec. Report No. 296, M.I.T. Lincoln Laboratory.

Wall, H.M., 1964. "Using a computer for circuit analysis", Electronics, November 2, pp. 56-61.

Warfield, J.N., 1963. Principles of Logic Design, Ginn And Co., Boston.

APPENDIX

UNIVERSITY OF ALBERTA

GRAPHICAL DISPLAY SYSTEM

The graphical display system at the University of Alberta consists of an IBM 360/67 linked to a CDC Graphical Remote Interactive Display (GRID) (CDC 1968). A data link is necessary to connect the GRID with the S/360. A half-duplex telecommunications line of 40.8 Kilobits/sec. having W. E. type 301B modems is used. To support the telecommunications, an IBM 2701 Data Adaptor Unit is used, and it in turn is linked to the S/360 via an IBM 2870 Multiplexor Channel. The 2701 is configured to support Binary Synchronous Communication.

GRID comprises:

(a) A display composed of a 21 inch electromagnetic deflection CRT. Display area is 12 inches x 12 inches, with 1024 x 1024 addressable locations. The spot size is 0.030 inches (maximum) and the beam positioning accuracy is ±1%. Display refresh rate is 50 hertz.

(b) A display controller (a modified CDC 160 A computer) which has two modes of operation: (1) display mode; (2) processor mode. Both of these modes share the same arithmetic and memory circuitry so they cannot operate simultaneously. The processor execution time is 2.4 to 9.2 microseconds. The display rates are as

follows:

Random point plot	- 4.0 to 12.0 microseconds
Incremental point plot	- 4.0 microseconds
Random symbols	- 5.2 to 18.4 microseconds
Incremental symbols	- 5.2 to 8.4 microseconds
Tabular symbols	- 2.8 to 8.4 microseconds

(c) Magnetic core composed of 4096 - 12 bit storage words. Relative, direct, and indirect addressing is available. The cycle time is 1.2 microseconds and access time is 0.8 microseconds.

(d) An operator control panel for off-line program entry, program checkout, and general maintenance.

(e) An alphanumeric and function keyboard. The alphanumeric portion includes the alphabet set, digits and 25 special characters. The function keyboard includes special edit keys, 10 function keys, an interrupt key, and 4 status switches (16 combinations).

(f) A light pen which can operate in capture or track mode.

(g) A Teletype ASR-33 with transfer rate of 110 bits/sec.